NETWORK MANAGEMENT USING ACTIVE NETWORKS

By

Antonio Canales Rivas

De Montfort University

March 2006

Approved by _____
Chairperson of Supervisory Committee

_____

_____

Program Authorized
to Offer Degree _____

Date _____

DE MONTFORT UNIVERSITY

ABSTRACT

NETWORK MANAGEMENT USING ACTIVE NETWORKS

by Antonio Canales Rivas

Chairperson of the Supervisory Committee:Professor Amelia Platt
Department of Computer Science

The main goal of network management systems is to ensure the quality of the services that networked elements provide. The management of a network involves co-ordinating and responding to alarms, performance indicators, traffic and accounting statistics and various other pieces of information which are needed to keep the network operating efficiently. The current network management systems suffers from the difficulty of integrating new technologies and standards into the shared network infrastructure, poor performance and complexity in accommodating new services in the existing architectural model. Active networks represent a new approach to network architecture. Active networks visualise the network as a collection of active nodes that can perform any computations, and a collection of active packets that carry code and are indeed programs. Active Networks will provide that the functions of the network nodes will not be longer be rigidly built-in by vendors who must follow designs dictated by slow and intractable standards committees. Also, network integrity will not be vulnerable against various ad hoc approaches toward network programming, as is the case today. This thesis will present the application of Active Networks technology as a solution to the current network management problems.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLES

Number *Page*

ACKNOWLEDGMENTS

The author wishes to give thanks to my family, especially to my dear wife Maria for her tolerance, my adorable mother for her support, my children Alvaro and Miriam for all the time borrowed and to my Supervisor Dr. Amelia Platt for her incredible patience and support.

GLOSSARY

ANTS: Active Networks Toolkit System.

ASE: Application Service Elements.

ASN.1: Abstract Syntax Notation 1.

ATM: Asynchronous Transfer Mode.

BER: Basic Encoding Rules.

CCITT: Comité Consultatif International Téléphonique et Télégraphique.

CIM/WBEM: Common Information Model/Web Based Enterprise Management.

CMIP: Common Management Information Protocol.

CMIS: Common Management Information Service.

CMISE: Common Management Information Service Element.

EGP: Exterior Gateway Protocol.

FTAM: File Transfer Access and Management Protocol.

FDDI: Fiber Distributed Data Interface.

HTTPS: Hyper Text Transfer Protocol Secured.

IAB: Internet Architecture Board.

ICMP: Internet Control Message Protocol.

IETF: Internet Engineering Task Force.

IIOP: Internet Inter-ORB Protocol.

ISDN: Integrated Services Digital Network

ISO: International Organization of Standardization.

ITU: International Telecommunication Union.

JAAS: Java Authentication and Authorisation Service.

JDMK: Java Dynamic Management Toolkit.

JMX: Java Management Extension.

LAN: Local Area Network.

MIB: Management Information Base.

MIM: Management Information Model.

MOC: Managed Object Class.

OSI: Open System Interconnection.

RMI: Remote Method Invocation.

SAME: Systems Management Application Entity.

SMI: Structure of Management Information.

SNA: System Network Architecture.

SNMP: Simple Network Management Protocol.

SSL: Secure Sockets Layer.

SS7: Signalling System No. 7.

TMN: Telecommunication Management network

TTL: Time to Live.

Chapter 1

INTRODUCTION

1.1 INTRODUCTION TO, AND MOTIVATION FOR, THE RESEARCH

1.1.1 THE WEAKNESSES OF EXISTING NETWORK MANAGEMENT STANDARDS

Communication networks are not viable if they cannot be managed properly. In general, communication networks comprise different devices, such as computers, switches and routers, which are interconnected by some physical connection media and communicate using standardised communication protocols. The management of a network involves co-ordinating and responding to alarms, performance indicators, traffic, accounting, security and various other pieces of information which are needed to keep the network operating efficiently. This is achieved by having a manager (management centre) routinely poll the managed devices.

This technique has served us well in the past, but over the last 10-15 years networks have increased in size and network devices are more diverse and complex. Furthermore networks are now expected to carry data from a much wider range of services. Thus, managers are now inundated with large amounts of information. This information is very often redundant, as the packets that arrive may simply report that there is no change in the state of the managed device. Also, in the event of problems in the network, the round-trip time that is needed for the information to reach the manager and the time for the subsequent

reply to reach the managed device is sometimes significant and may be too late for appropriate and timely action to be taken.

There are a number of general problems, shared by all network management standards, which are considered to be key problems which must be solved before contemporary networks can be managed effectively and efficiently. These are as follows.

- Network management systems are centralised and therefore not scalable. This problem is partially overcome by implementing administrative decentralisation of the managers, but the intelligence is still centralised.
- The current infrastructure is too static. For example, to install a new central management application is straightforward but to upgrade all managed devices is not. To do that it is necessary to upgrade every device before the new network management application can be made available. Maintenance is therefore expensive and time consuming.
- This centralised paradigm means that non-trivial tasks are often carried out in an inefficient way because they typically require a number of messages to be exchanged between the manager and the device. This is a particular problem with time-critical tasks. For instance, if the manager is required to carry out a statistical estimate of the number of active connections in a group of devices, a number of messages must be exchanged and this takes time. Since connections are continually being established and taken down, real-time monitoring is problematic with the existing paradigm.

- Centralised management is prone to the bottlenecks that can impede communication between the manager and the network devices. This is especially evident when a device error occurs since management traffic will then increase at the very time when the network is least able to cope with it. Similarly, if the manager fails, devices can do nothing to solve the problem since they are only programmed to respond to the central manager commands. Potentially therefore, a small problem may lead to complete system failure.

- Inevitably, the approval for updating standards lags well behind technology developments. There needs to be a way of reducing this time lag.

- Skilled personnel are often needed to collate and analyse data values held in MIBs. Since the manufacturers are continually defining new proprietary MIB variables, this analysis work is becoming more and more specialised. In contrast, many MIB variables are seldom used because the management applications, which originally used them, have been superseded by upgrades. A more dynamic MIB maintenance procedure is required.

- Existing management systems do not allow the creation of an external view of MIB data. Much of the data held in the MIB is related, but these relationships cannot be formalised within the device and then extracted by a manager.

- When multiple managers are used, it is not easy to share results between them. For example, one manager may create a set of objects for filtering and correlating different MIB values. Other managers could benefit from having access to these objects but current standards do not allow this.

- The centralised paradigm does not permit the creation of a flexible set of data filters inside a managed device. Indeed, the Internet management system specifically excludes filters and the OSI system has only a limited facility. Increasingly, this is becoming a serious disadvantage.

- The lack of a flexible security system. In network management, each device controls its local MIB and it should answer the requests from a number of managers. Current network management security systems comprise a standardised system of encryption that must be used in all circumstances. This is clearly inadequate.

- The implementation of network management systems is often (unintentionally) not compliant with the standards. For example, experiments at Bell labs to test various SNMP-managed bridge products [7] showed that some bridges were able to provide an SNMP manager with accurate and consistent counts of packets, while others were not. In a second experiment to test various SNMP-managed IP routers [8], not one single vendor returned accurate values for the five counters in the Interfaces group of MIB variables. Despite the emphasis on simplicity in SNMP, products from reputable manufacturers interpreted some of the most basic network management parameters differently. Other management systems, CMIP for example, are more complex.

- There is a large volume of potentially relevant management data, which is not available in any MIB. This means that the standardised structure of a MIB does not meet the needs of modern network management systems. What is needed is a

system for creating new MIB variables, on demand. This problem was illustrated by Hasan [9] who showed that there is no procedure for measuring flow between different hosts using the MIB, and this leads to under-utilisation of the network.

The problems noted above have arisen because existing network management standards have not evolved to cope with the changes in networks. Indeed there have been few changes to network management standards since their inception. This is because the process of changing network management standards is lengthy and difficult (the slow introduction of IPv6 is one example). Backward compatibility is also essential if legacy devices are to be managed alongside new technologies. Furthermore, incremental deployment is needed to allow new services to be deployed.

1.1.2 Active Networks as a solution to the problems of Network Management

A possible solution to the various network management problems and their constraints is the use of a new technology called Active Networks. Active Networks technology has been proposed as a way in which services can be introduced into the network. Active Networks exploits mobile code together with a programmable infrastructure to provide rapid and specialized service introduction. Active Networks are fundamentally different from the networks in use today. Traditional networks rely on prior agreement of message processing between communicating parties, typically with sharp distinction between the roles of intermediate and end-systems. In contrast, Active Networks rely on prior agreement of a computational model. An Active Network is a virtual network that allows Active nodes (enhanced network devices) to perform computations defined by Active Networks services.

The code for these services travel inside network packets and are executed in Active nodes resulting, in the modification of the state and behaviour of the Active Node.

The purpose of this research is to answer the question:

Can Active Networks solve all the documented problems of Network Management Systems?

1.2 AIMS OF THE RESEARCH

1.2.1 TO PRODUCE A THOROUGH ANALYSIS OF THE STRENGTHS AND WEAKNESSES OF SNMP

To produce a report which identifies the strengths and weaknesses of SNMP from published sources.

1.2.2 TO CONSTRUCT A FRAMEWORK WHICH WILL ALLOW A COMPARISON TO BE MADE BETWEEN STANDARD SNMP SERVICES AND ACTIVE SNMP SERVICES

To make a comparison of standard SNMP services and Active SNMP services a test-bed must be designed and built. This is the SNMP Active Networks Toolkit. The Toolkit functionality is divided in two parts:

- Standard SNMP Network Management System
- Active SNMP Network Management System

This first part will provide a complete SNMP system. The Java Management Extension standard (JMX) has been used to build this system. The second part will integrate Active Network technology to the Standard SNMP Network Management System to create the Active SNMP Network Management Toolkit. The Active Network Transport System

(ANTS) which has been developed at MIT [10, 11] was the Active Technology used. The ANTS toolkit is freely available for experimentation and is written entirely in Java.

1.2.3 To design and build a set of services which allow a comparison to be made between standard SNMP services and Active SNMP services

A set of key services will be chosen to evaluate the relative merits of standard SNMP services and the corresponding Active SNMP services. The evaluation procedures will be based on a comparison of quantitative metrics (e.g. bandwidth requirements, delay) and more qualitative metrics (e.g. flexibility and functionality) of the services.

1.2.4 To provide an ANTS system to dynamically load extensions in the Active Nodes

ANTS currently does not provide the functionality to deploy extension code automatically; this must be done statically. This is considered a useful feature, therefore the aim is to enhance ANTS with this functionality.

1.2.5 To provide an ANTS system to dynamically load Active Network topology

Another useful enhancement for ANTS is to have a system where the dynamic nature of an Active Networks topology can be captured. An Active Network can be deployed incrementally, therefore its topology inevitably changes. An Active Service must adapt its operation to the changing topology. Jini, a Java distributed system, was selected as the basis to provide this ANTS enhancement.

1.3 CONTRIBUTION TO KNOWLEDGE

A brief summary of the contribution to knowledge is:-

1. The analysis of all documented SNMP problems.
2. The design and implementation of a framework which provides a standard SNMP prototype network management system and an Active Networks management system.
3. The design of a number of key generic Active services, each of which provide the design pattern and framework to solve multiple SNMP problems.
4. The design and implementation of a system for dynamically loading extension code in ANTS.
5. The design and implementation of a system which maintains (dynamically) the ANTS network topology.

1.4 OUTLINE OF THE THESIS

Chapter 2 is concerned with Network Management issues. An overview of network management is given together with an overview of the main network management standards. It is argued that SNMP is considered the de facto network management standard and thus SNMP is selected as the network management standard bench mark for this research.

However, although SNMP is the most widely used network management standard, nevertheless it has many deficiencies which have been reported over a number of years. There are now 3 versions of SNMP and each version has tried to address some of

deficiencies, with varying degrees of success. A comprehensive analysis of the strengths and weaknesses of SNMP is presented in Chapter 2.

Active Networks is the subject of Chapter 3. A brief overview of what Active Networks are and the advantages they offer is presented. There are a number of Active Networks technologies, some of which were designed with a particular problem domain in mind. A number of the more widely used technologies are presented in more detail. It is argued that ANTS is the most suitable technology for network management projects and thus this is chosen as the technology for this research. The key network services which have been chosen to be part of the Toolkit are also identified in this Chapter.

One of the aims of the research is to build a SNMP Toolkit. The environment for this toolkit is the subject of Chapter 4. The key Java technologies used to provide the environment are JMX, JDMK and Jini and these are discussed in detail. Chapter 5 then presents a detailed design of the architecture for the standard SNMP Network Management system and this is then used to build the system. This is then enhanced to provide a second system – the Active SNMP network management system.

The final part of the Toolkit is presented in Chapter 6. This discusses the design of each of the key Active Networks services selected for investigation in Chapter 3. The design, analysis and evaluation of each of the services are presented together with a comparison of the strengths and weaknesses of the standard SNMP version and the corresponding Active SNMP version. Finally, Chapter 7 provides a critical review of the research and discusses further work.

1.5 CONSTRAINTS

The Toolkit produced as part of this research is based on a number of Java technologies, therefore requires a Java platform.

The Toolkit, is a combined real SNMP network management system and an Active SNMP network management system. The SNMP network management system is complete in that all the SNMP commands are implemented, and the MIB has all the variables defined. However, the network used as the prototype for this research had only host computers as network devices and thus in this prototype not all the MIB variables have values. Similarly, the Active SNMP network management system is complete, in that it has the full standard SNMP functionality. Furthermore, it has a number of new complex services (e.g. Macro service) provided through the use of the ANTS technology. It is expected that further Active Services will be designed and implemented according to the needs of the network manager.

Chapter 2

NETWORK MANAGEMENT

2.1 INTRODUCTION

This Chapter discusses network management, including the major network management standards in detail. Section 2.2 gives an introduction to network management and introduces terms used to define the various elements in network management. The major network standards are described in detail in Section 2.3. In it argued in Section 2.3 that SNMP is the de facto network management standard and thus this is chosen as the standard for this research.

Section 2.4 presents the strengths and weaknesses of SNMP although many of these weaknesses also exist in the other standards. The weaknesses are analysed in detail and it can be concluded that they fall into 2 main categories, Network Protocol Issues and Architecture Issues. Chapter 3 uses this analysis to decide which services are the most appropriate to answer the research question posed in Chapter1.

2.2 INTRODUCTION TO NETWORK MANAGEMENT

Network management is the action of initializing, monitoring and modifying network functions. This implies that management should first initialize the various network systems (configuration management). If no errors are made, the network comes into service and the operational phase starts. During this phase, management monitors the various network

systems to check if no errors occur. In case of failures, malfunctioning systems will be identified, isolated and repaired (fault management). Monitoring the network is also useful to detect, for instance, changes in the traffic flow. Once such changes are detected, network parameters may be modified to optimize the network performance (performance management). Security of the network is also an important function and becomes increasingly more critical (security management). Also, it is usual to charge for the network resources used (accounting management).

An important goal of network management is to support an integrated approach to the management of a network. This is the key goal of network management standards: to develop an integrated set of procedures and standards that apply equally well across different vendors and networks.

The main elements in a network management system are:

- Manager: This is typically a software application and its main function is to direct the operation of the agents. The manager is usually deployed in a special device called a Manager Console.

- Agent: It is typically a small software application deployed in all managed network devices. Its main function is to respond to commands from the manager and signals from the device software and hardware. The agent effectively sits between the hardware and the manager.

- Management Information Base (MIB): This is found in the agent. The agent stores information from the software and hardware in the MIB and returns MIB

information requested by the manager. The manager also has knowledge of the MIB structure.

In the simplest sense, a network management system really contains nothing more than protocols that convey information about networks devices back and forth between various agents in the system and the manager. These elements are not required to be placed at a particular location in the network. In current implementations, the agent software is usually placed in network devices such as servers, gateways, bridges and routers and hosts (e.g. user PCs).

These agents represent the management support functionality through which manager(s) initialize, monitor and modify the network devices and their behaviour. Compared to managers, agents are usually simple applications. To allow managers to communicate with their agents, a communication protocol is required. Examples of such protocols are CMIP and SNMP.

One of the most important elements in network management is the MIB. The MIB is a collection of static and dynamic management information. The MIB is an abstraction: the actual mechanisms for storing and accessing its contents are not explicitly defined. Each management system builds its own schema, access mechanisms, and storage management facilities. Management systems must protect the MIB contents in case of management system failures and other types of disruptions.

The information kept inside the MIB can be catalogued in two categories:

- Static. The information does not change. E.g., information about the location of network devices, warranties, serial numbers, and software versions are various examples of this category.
- Dynamic. The information changes frequently over time. Examples of dynamic MIB variables are:
    - Counters.
    - Gauges. Describe operations that can change in more than one direction. E.g., a gauge is used to monitor the server CPU utilization.
    - Threshold. Used to detect situations that require special attention. A gauge may have several thresholds associated with it, for exceeding a maximum value or falling under a minimum value.

The following sections will provide details about the main network management standards currently used, with particular emphasis on the Simple Network Management Protocol (SNMP) standard. An analysis of the strengths and weaknesses found in SNMP is provided.

2.3 NETWORK MANAGEMENT STANDARDS

There are several organizations that have developed services, protocols and architectures for network management. The three most important organizations are:

- The International Organization of Standardization (ISO).

- The Telecommunication Standardization Sector of the International Telecommunication Union (ITU).
- The Internet Engineering Task Force (IETF).

ISO was the first organization[1] to develop an architecture for network management, as part of its 'Open System Interconnection' (OSI) program. The first proposals for such an architecture appeared during the early 1980's; nowadays a large number of standards exist for the architecture as well as for network management services and protocols. Of these standards the 'OSI Management Framework' and the 'Common Management Information Protocol' (CMIP) are probably the best known examples.

The development of management standards for telecom networks was left to CCITT (now named ITU-T). In 1985 the CCITT started the development of such management standards; these standards have become known as the 'Telecommunications Management Network' (TMN) recommendations. TMN has integrated many OSI Management concepts inside its architecture.

Initially in the Internet, the Internet Architecture Board (IAB) intended to apply the OSI management approach, but at the time the size of the Internet reached a level at which management became essential. Meanwhile OSI management groups were still busy discussing the OSI management framework. Since implementations of OSI management standards were not expected to appear soon, the IAB requested the IETF (the organization

[1] The bibliography for the network management standards considered in sections 2.2 and 2.5 can be found in the references [1],[3],[5], [99],[100],[101],[102],[103]

who is responsible for the development of Internet Protocols) to define a management standard. This 'Simple Network Management Protocol' (SNMP) was completed quickly and soon many manufacturers started the production of SNMP compliant Systems.[2] Although SNMP has many deficiencies, it has become the de facto standard for the management of data communication networks.

The next section will provide a brief examination of the network management standards proposed by ISO and ITU-T organizations, followed with a more detailed analysis of the SNMP standard.

2.3.1 OSI MANAGEMENT FRAMEWORK

Although the origin of OSI management can be found in ISO, most of the work was performed in collaboration with the ITU-T. The standards that result from this cooperation are published by both organizations without technical differences. Within the ITU-T, the OSI management recommendations are published as part of the X.700 series.

There are five functional areas into the OSI Management Framework. These are:

- Fault Management.
- Configuration management.
- Accounting management.
- Performance management.
- Security management.

---

[2] Note, SNMP is used to identify both a network management standard and also a network management protocol.

The OSI network management standard uses many of the Object Oriented Development concepts inside its architecture. The resources that are supervised and controlled by network management are called managed objects. Management information is the information associated with a managed object that is operated on by the OSI Management protocol to control and monitor that object. A managed object is completely described and defined by four aspects of the OSI network management standard:

- Its attributes (characteristics or properties), that are known at its interface (visible boundary).
- The operations that may be performed on it.
- The notifications (reports) it is allowed to make.
- Its behaviours, exhibited in response to operations performed on it.

2.3.1.1 OSI Systems Management Overview

Systems management is used to manage an entire OSI system. It provides mechanisms to manage multiple OSI layers and is accomplished through application layer protocols. The ISO and ITU-T have developed a presentation and transfer syntax to be used by application layer protocols. The ISO standard is titled Abstract Syntax Notation 1 (ASN.1). In addition, the Basic Encoding Rules (BER) provides a set of rules to develop an unambiguous bit level description of data. In summary, ASN.1 describes an abstract syntax for data types and values, and BER describes the actual representation of the data. The information aspects of the systems management model deal with the resources that are being managed.

These resources, as it was indicated above, are viewed as 'managed objects'. The Management Information Base (MIB) identifies the managed objects to be managed. The OSI has published its own 'MIB', but does not use this term, because the OSI approach is somewhat different. The term management information model (MIM) is used to describe the same type of functions that are covered in the Internet MIB. OSI uses templates to define a managed object class (MOC). MOCs provide a convenient means to group related resources together. This means that it is possible to encapsulate (or contain) objects within other objects and, in so doing, invoke operations or receive notifications only on the relevant 'layer' of the encapsulated objects. The template describes aspects of the MOC such as attributes, and syntax. The MOC template can reference other templates, so the reference allows the template to 'import' other templates to the MOC template.

OSI systems management is organized in a centralized fashion. According to this scheme, a single manager may control several agents. The manager performs operations upon the agents and agents forward notifications to their managers. Besides, the OSI management environment may be partitioned into a number of management domains. The partitioning can be based on functional requirements (e.g. security, accounting and fault management), but also on other requirements (e.g. geographical and technological).

In the operational OSI network management, an application-entity called Systems Management Application Entity (SMAE) is involved. This entity is responsible for implementing the OSI System Management activities. A SMAE is a collection of cooperating Application Service Elements (ASEs).

Into a SMAE there is a special ASE called a Systems Management ASE (SMASE), which creates and uses the Management Protocol Data Units (MPDUs) that are transferred between the management processes of two machines. The SMASE may use the communications services of one Common Management Information Service Element (CMISE) or to use another ASE.

OSI has defined the Common Management Information Service (CMIS) as the preferred service for the exchange of management information. CMIS role is restricted to the transfer of management information. Basically CMIS consists of a service definition and a protocol specification to support the services. This protocol is referred as Common Management Information Protocol (CMIP) and is used to transport network management information between agents and managing processes.

2.3.1.2 OSI Systems Management weaknesses

OSI management is faced with several problems:

- The OSI management approach uses implicitly the layer protocols that are being managed for the exchange of management information too. The problem with this dependence is that should a fault exist in a protocol implementation it may not be possible to convey this fault to the manager.
- OSI management explains how individual management operations, such as GETs and SETs, should be performed, but not the sequence in which these operations should be performed to solve specific management problems.

- OSI management is rather complicated and the standardization took much longer than expected.
- Although most manufacturers declared their support for OSI management, only a few offer implementations.
- Management systems that are based on the OSI architecture are presently more expensive that management systems that are based on the Internet management architecture (SNMP).

Due to these problems, it is questionable whether OSI management can dispute the SNMP dominant market position.

2.3.2 TMN MANAGEMENT FRAMEWORK

A TMN (Telecommunications Management Network) is conceptually a separate network that interfaces a telecommunications network at several different points. The interface points between the TMN and the telecommunication network are realised by exchanges and transmission systems. For the purpose of management, these exchanges and transmission systems are connected via a data communication network to one or more network management systems. The network management systems perform most of the management functions. Initially there was little collaboration between the management groups of ITU-T and ISO. But in the last revised version, the collaboration between ITU-T and ISO was improved. This resulted in the inclusion of many OSI management ideas into TMN. The most important changes to TMN were: the manager-agent concept, the new object oriented

approach and the idea of management domains were included. Despite this cooperation, fundamental differences in philosophy still exist.

An interesting difference between OSI and TMN management is that OSI has defined a single management architecture whereas TMN defined multiple architectures at different levels of abstraction, (Functional architecture, Physical architecture). These multiple architectures have multiple levels of management responsibility. The structure is known as the responsibility model. The advantage of having such structure is that it becomes easier to understand and distinguish the various management responsibilities.

Finally, TMN suggests a conceptual separation between the network that is managed (the telecommunication network) and the network that transfers the management information (the data communication network). This is comparable to the proposal to introduce a separate network to exchange signalling information. In this sense TMN resembles SS7 networks.

2.3.2.1 TMN Systems Management Overview

TMN uses the conventional manager/agent concept for the exchange of messages between functions: the manager issues management messages and receives notifications, and the agent manages the managed objects, receives messages (directions) from the manager, and emits notifications to the manager.

To realize its purpose, TMN functional architecture is defined in terms of function blocks and reference points. Function blocks contain functional components (such as Presentation functions or MIBs) and may be compared to OSI protocol entities. Reference points are

used to interconnect function blocks and may in OSI terminology be compared to underlying management information service providers. TMN function blocks are divided into two major areas:

- Data communications functions (DCF). The DCF provides layers 1 to 3 of the OSI model.
- Message communications functions (MCF). It consists of protocol stacks that allow the communication of the function blocks to the Data Communication Functions.

The most elementary form of TMN functional requirements may be viewed as a cooperative activity involving two application processes, one in a managing system and the other in the managed system. In architectural terms, each application process may be equated to a management application function in a TMN function block.

Information models are the essence of TMN standards and are categorized by the following characteristics:

- Whether they are specific to the management of one type of telecom technology (e.g., SDH), or may be applied to manage more than one.
- Whether they represent abstractions of network resources (physical of logical), or specific aspects of management applications.
- Whether they are concerned with element, network, service, or business management, or are applicable to more than one type of management.

TMN information modelling has been extended to cover switching, transport (PDH, SDH, ATM), and wireless network resources, primarily from an element and network

management perspective, while applications include configuration, fault, security, performance, and accounting management.

TMN communication protocols have come in a variety of packages to support the diverse needs of its users categorized from two perspectives: either on the basis of the data network type employed or by the application. Currently support is provided for X.25 packet, 802.3 LAN, ISDN, Signalling System No. 7, and TCP/IP-based networks. Application protocol types supported include transaction class using CMIP, file transfer class using FTAM, and directory service class using X.500.

As it has been shown, TMN is particularly aimed at the management of communication networks, where the introduction of additional equipment and transmission systems required for the management network is needed. This is due to the lack of adequate facilities to transfer management information.

2.3.3 THE SNMP STANDARD

The Simple Network Management Protocol is the de facto worldwide standard for managing heterogeneous networks. The main network sector for SNMP are LANs but it is now used to manage SNA (IBM's System Network Architecture), Frame Relay, and ATM networks. Even the telecommunication carriers have introduced SNMP management interfaces for their network services. Rapid evolution and aggressive inclusion of new MIB definitions has kept SNMP moving forward in terms of its ability to manage more complex and varied environments.

The original version of SNMP (now known as SNMPv1) rapidly became the most widely used vendor-independent network management system. However, as SNMP gained widespread acceptance, its deficiencies became apparent. These include a lack of manager-to-manager communication, the inability to do efficient bulk data transfer, and a lack of security. All of these deficiencies were addressed in SNMPv2, issued as a set of proposed Internet standards in 1993.

SNMPv2 also supports a bulk transfer feature, making the transfer of large amounts of information much more effective than with SNMPv1. SNMPv2 also added security mechanisms, replacing the simple community string idea of SNMPv1 (basically an open password).

SNMPv2 has not received the acceptance its designers anticipated. While the functional enhancements have been welcome, developers found the security facility for SNMPv2 too complex. The result has been one minor success and one failure. But in 1996, SNMPv2 progressed from proposed to draft Internet standard status. Then, in 1997, work began on SNMPv3, which makes additional minor functional changes and incorporates a new security approach.

SNMP has proven itself as an open, interoperable standard with hundreds of vendors offering products. However, interoperability suffers from the proliferation of proprietary extensions to the standardized MIB. For example, there are MIBs for monitoring and managing DECnet Phase IV Networks, SNA Network Addressable Units, Frame Relay, and ATM networks. These proprietary extensions, while adding extra value to management solutions, also lock customers into the vendor.

2.3.3.1 SNMP Network Management Architecture

The model of network management that is used for SNMP includes the following key elements:

- Management station
- Management agent
- Management information base
- Network management protocol

Only the last two elements are the subject of SNMP standardization.

2.3.3.2 Management Information Base

To identify all variables that can be managed in the Internet, a large number of Management Information Base (MIB) standards have been developed. In addition to these standards, one special standard exists to define how MIB variables must be described. This standard is called the Structure of Management Information (SMI). It defines, for instance, the subset of ASN.1 constructs that can be used to describe management variables.

To ensure the unique identification of each management variable, the SMI introduces the concept of a naming tree. The leaves of this tree represent the actual management information. The Management Information Base for Network Management of TCP/IP-Based Internets (MIB-II) is the most important and probably best known MIB; it contains all the variables to control the major Internet protocols (e.g. IP, ICMP, UDP, TCP, EGP

and SNMP). The structure of this MIB is simple: all management variables that belong to the same protocol are grouped together. The groups are: System, Interfaces, Transmission, Address Translation, IP, ICMP, TCP, SNMP, UDP and EGP. As the group names suggest the 'groups' information is related mainly to the main Internet network protocols.

The Internet MIB clusters the managed objects within a group into tables. The table is organized as a two-dimensional matrix where the objects form the columns of the table. Each instance of an object forms a row entry into the table. As an example, the Interfaces group will be analyzed in more detail. This group is mandatory for all systems. The purpose of this group is to provide information on an object's interface(s). It describes the type of interface, such as SDLC or Ethernet, etc., and provides statistics on the operations occurring at the interface. It consists of one individual entry and a 22-column table which is shown Table 1

ifNumber	A value containing the number of network interfaces at this system.
ifIndex	An unambiguous value for each interface, its value can range from 1 to the value of ifNumber.
ifDescr	Textual information describing the interface, typically a product name, a manufacturer name, and a version number of the interface.
ifType	Identifies the specific type of interface, and currently contains values to identify one of 32 interface types such as Ethernet, SDLC,Fiber Distributed Data Interface(FDDI),etc.
ifMtu	The maximum size of the protocol data unit (PDU) that can be serviced at this interface, specified in octets.
ifSpeed	In bits per second, the value of the interface's speed capacity, if relevant.
ifPhysAddress	The interface address immediately below the network layer in a typical protocol stack, typically an IEEE MAC address for a LAN interface.
ifAdminStatus	The desired state of the interface with values: up=1, down=2, testing=3. The testing state indicates that no operational packets can be passed.
ifOperStatus.	The current operational state of the interface with values: up=1, down=2, testing=3. The testing state indicates that no operational packets can be passed.
ifLastChange	In time ticks, the time the interface entered its current operational state.
ifInOctects	Total number of octets received at this interface since the last initialization.
ifUcastPkts	The number of packets delivered to the next layer protocol, no broadcast packets only.
ifInNUcastPkts	The number of broadcast or multicast PDUs delivered to the next higher-layer protocol.
ifInDiscards	The number of PDUs discarded at this interface and not delivered to a higher-layer protocol. No errors occurred but the PDUs were discarded for other reasons (buffer space problems for example)
ifInErrors	Number of PDUs that contained errors, were ill formed, unintelligible, etc. (not delivered to the next higher-level protocol).
ifInUnknownProtos	The number of PDUs discarded because the PDU is related to an unsupported or unknown protocol.
ifOutOctets	Total number of transmitted octets at this interface since the last initialization.
ifOutUcastPkts	Total number of PDUs that the next higher-layer protocol requested this interface to transmit to a non-multicast or no broadcast address. This value includes those PDUs which may have been discarded or otherwise not sent.
ifOutDiscards	Total number of outgoing PDUs which were discarded even though they contained no errors (again, for example, buffer space problems).
ifOutErrors	The total number of outgoing PDUs that contained errors and were discarded.
ifOutQLen	The total length of the output queue in total number of PDUs involved.
ifSpecific	A reference to a MIB definition that relates to the media used at the interface. This is used to provide additional information and may possibly not contain a value.

Table 1. MIB-II Interfaces Group

Each row of the table is a MIB variable relating to one physical interface.

Next to the standardized MIBs there are also a large number of enterprise specific MIBs. Together these MIBs define more than twenty thousand MIB variables. Unfortunately, no clear structure has been developed to explain the relationship between these MIBs.

Examples of MIBs are: IEEE 802.5 (RFC 1231), Remote Network Monitoring (RFC 1271), X.25 (RFC 1461), Network Services Monitoring (RFC 1565), Mail Monitoring (RFC 1566), X.500 Directory Monitoring (RFC 1567), and ATM (RFC 1695).

2.3.3.3 SNMP Network Management Protocol

SNMP was designed to be an application-level protocol that is part of the TCP/IP protocol suite. SNMP typically operates over the user datagram protocol (UDP), although it may also operate over TCP. For a standalone management station, an agent controls the MIB and provides an interface to the network manager.

Each agent must also implement SNMP. In addition, the agent's responsibility is to interpret the SNMP commands and control the remote access to the MIB.

From a management station, the manager can issue four types of SNMP commands: GetRequest, GetNextRequest, GetBulk and SetRequest. The first two commands are variations of the GetRequest command. All three commands are acknowledged by the agent in the form of a GetResponse message, which is returned to the manager. In addition, an agent may issue a Trap command in response to an event that affects the MIB and the

underlying managed resources.SNMPv2 protocol included a special command to obtain MIB tables. This command was called GetBulk.

SNMP operates over UDP, which is an unreliable (connectionless) transport protocol, and SNMP is itself connectionless. Thus, no connections are maintained between a manager and its agents.

If a management station is responsible for a large number of agents, it becomes impractical to regularly poll all agents. Instead, each agent is responsible for notifying the manager of any unusual event. Examples are if the agent crashes or is rebooted. These events are communicated in SNMP commands known as Traps.

2.4 SNMP STRENGTHS AND WEAKNESSES

The SNMP has won the short-term battle for supremacy: it has succeeded spectacularly and is the de facto worldwide standard for network management. However, as the protocol gained widespread acceptance, its deficiencies became apparent. The following sections will provide details about the strengths and weaknesses of SNMP.

2.4.1 SNMP STRENGTHS

It would difficult to find any LAN devices (hubs, bridges, routers, and switches) that do not offer an SNMP agent and management tool. SNMP has spread to other environments such as SNA (IBM's System Network Architecture), Frame Relay, and ATM. Even the telecommunications carriers have introduced SNMP management interfaces for their

network services. Rapid inclusion of new MIB definitions has kept SNMP moving forward in terms of its ability to manage more complex and different environments.

The design goals of SNMP were focused upon the creation of simple and cheap agents that offered the following advantages:

- The simplicity brings few opportunities for failure.
- The cheapness attracts many vendors, because a low investment is sufficient to enter to the market.
- The resources in the agent devices are modest, thereby the impact, on performance and throughput, of using SNMP is minimal.

2.4.2 SNMP WEAKNESSES

Despite its success, SNMP has a significant number of weak points that must be addressed if it wants to maintain its supremacy in the network management area. The weaknesses of SNMP can be divided in two categories:

- Network Protocol Issues.
- Architectural Design Issues.

Table 2 presents a summary of the weaknesses and these are discussed in detail in the remainder of the section.

Network Protocol Issues	UDP limitations.
	Inefficient protocols characteristics.
Architecture Design Issues	Inefficient use of network management functions from applications.
	Performance limitations of polling.
	Dynamic MIB views.
	Data Sharing among managers.
	A flexible plug in security system.
	Lack of features to provide auto topology.
	Lack of semantic data in the MIB variables.
	Lack of a flexible system for defining generic event traps.
	Lack of facilities to provide metadata inside the MIB.
	Lack of facilities to reset the system to a default state.
	Lack of mechanisms to support the concept of time in the MIB data.
	Lack of mechanisms to support adding new MIBs dynamically to a system.
	Lack of scalability to manage a bigger system.
	Lack of flexibility to update the system.

Table 2. Summary of the weaknesses in SNMP.

2.4.2.1 Network Protocol Issues

2.4.2.1.1 UDP limitations

There are several limitations in the use of UDP as the transport protocol. First, UDP is not a reliable protocol, which means there is no guarantee that SNMP messages arrive safely. For example, to be sure that a SetRequest command has been executed requires a three-step process. First, the network manager must get the value of the object from the MIB using the GetRequest command. Then the manager must issue the SetRequest command and finally, it must reread (issue another GetRequest command) to check that the original value has been changed. [12]

Second, UDP limits the size of the amount of data that can fit into a single UDP packet (64 K bytes). This maximum size was a design goal, but for some kind of management operations, like GetBulk commands, it results in a large overall latency value for the transfer of large amounts of data. [13]

2.4.2.1.2 Inefficient protocols characteristics

Some features of SNMP are inefficient. One example is the way in which tables must be retrieved from the MIB. The essential problem is that while the structure of the table is known by the manager in advance of the retrieval, the size (number of rows currently stored) is not. The GetBulk command is used to retrieve tables. The GetBulk command requires the manager to specify the number of rows which must be retrieved (and the offset from the beginning of the table). If the table contains a large number of rows and the manager specifies a small number of rows to be retrieved then clearly only a fragment of

the table will be retrieved and further GetBulk command will be required. Conversely, if the table is small and the manager specifies a large number of rows to be retrieved then the entire table and the MIB variables stored after the table will be returned. Thus a large amount of redundant data will be retrieved unnecessarily. A further complication of spreading the table retrieval over a number of GetBulk commands is that the consistency of the table contents cannot be guaranteed; this is the classic database problem which is typically solved by using transactions. [14]

Another deficiency in SNMP is that there is no threshold concept whereby action is taken if a MIB variable exceeds some defined threshold value. Only systems using the RMON MIB with a probe device have this facility. [5]

Finally, there is no support for the manager to issue direct imperative commands to the agents. Instead, the manager must use the SetRequest command to set a value in the MIB and this value will trigger a response in the agent.

2.4.3 Architecture Design Issues

2.4.3.1 Inability of applications to benefit from network management information

Network management information and functions are entirely separated from user applications. It means that the applications are not aware of the existence of the network and therefore they cannot benefit from this knowledge. For example, a video streaming program could adapt its streaming algorithms based on network management information. This is likely to provide a better service to the user.

2.4.3.2 Performance limitations of polling

Network monitoring is one of the most significant functions carried out by network management. In SNMP, polling is used to get the information needed for monitoring the network. If the time interval for two consecutive polling requests is too long, then it is possible to miss errors. [15,16,17] Conversely, if it is too short, then the polling message traffic increases and thus imposes a heavy load on the network. Furthermore the probability of duplicate request increases. [8] The scale of these problems increase as the network size increases. The SNMP alternative to polling, the event-trap mechanism, is not adequate because only a few traps are defined and there is an interoperability problem when proprietary traps are used. Also, another problem with traps is its unacknowledged nature. [5] The performance issues relating to polling has recently been observed in the UK University intranet SuperJanet [18] where the amount of Network Management traffic affecting to the network performance.

2.4.3.3 Dynamic MIB views

The Current SNMP framework makes it difficult for a network administrator to choose a set of variables from the agents' MIB to create a personal view of the data. [19,20] This feature could be of great utility to the manager. [21,18] The concept could be extended further with the possibility of creating new MIB variables dynamically based on the combination of several other ones. It is known that that not all the information about management is in the MIB (examples are the use by managers of the programs Ping and Traceroute and the proposal of Remote Ping and remote Traceroute MIBs [22]) and not all the information in the MIB is used very frequently by the managers.

2.4.3.4 Data Sharing among managers

When multiple manager consoles are used, it is not possible to share information among them. Related to the concept of sharing information is the decentralisation of network management based on the use of the Inform command with the Manager-to-Manager MIB. The IETF are also trying to incorporate the sharing of information using a solution called Script MIB. [23] An alternative solution to the problem of sharing information is the creation of MIB variables dynamically by the managers (discussed in the previous paragraph). A manager could create a set of new MIB variables by filtering and correlating other MIB variables. These new MIB variables could then be shared with others managers, thereby avoiding the need for all managers repeating the same process. [24]

2.4.3.5 A flexible plug in security system

The latest version of the SNMP protocol (SNMP v3) permits the use of user security systems. [25] However, these security systems are set statically. A more flexible system would permit the dynamic plug-in of different user security systems depending of the changing circumstances of the users.

2.4.3.6 Lack of features to provide auto topology

The main goal of auto topology is that when a management station is installed, it will automatically learn the configuration of the entire network. [26] Knowledge of the up-to-date physical topology of an IP network is crucial to a number of critical network management tasks. There are several aspects to auto topology:

- The discovery of devices. Currently the strategies used are

- Send a broadcast SNMP packet.
- Sending an ICMP ECHO packet to every possible address on the network.
- The use of the capabilities of passive monitoring of a RMON probe. [27])

- The need for having a dynamic database, which is updated continuously with additions, replacements, or changes of network devices.
- The drawing of the logical network.
- The establishment of a hierarchy of the devices in the system.
- A set of criteria must be given to choose the devices that will be wanted most.

2.4.3.7 Lack of semantic data in the MIB variables

The SNMP protocol supports the storage and management of various information relating to the MIB variables. However there is no information about this information (metadata) that can be used to manage the individual devices and the network in an intelligent manner. [28,21] A system that enhances MIBs definitions with semantic data is needed. [30]

2.4.3.8 Lack of a flexible system for defining generic event traps

Currently, SNMP has only a limited number of standard traps defined. The current trend is to extend this by using specific enterprise traps, but this leads to a compatibility problem. [5]

2.4.3.9 Lack of facilities to provide metadata inside the MIB

There is currently no facility for storing representation information in the MIB. This information could consist of descriptive labels for use on a table or graph, formatting information, help text, etc. This would be useful to the applications, which will use the MIB data. This information would help the management station to provide a better user interface to SNMP data. [31]

2.4.3.10 Lack of facilities to reset the system to a default state

In the design of the standards MIBs (I and II), it was specified that there was no requirement to be able to reset the system to a default state and it was suggested that Uninterruptible Power Supply (UPS) technology could be used instead. [32] This is not suitable for all scenarios.

2.4.3.11 Lack of mechanisms to support the concept of time in the MIB data

To do a correct MIB data interpretation, a manager must be aware of having reliable and valid data. Actually the unit element related to time in the standard MIB is the scalar SysUpTime in the System group, which gives the time (in hundredths of a second) since the agent came up. [33] It is clear that this feature is not enough to provide consistency in the MIB data analyzed by the manager. [104] For example, in the long-term monitoring of a particular MIB variable, the time information is critical. If in this interval, the agent is rebooted, the SysUpTime will start to count again and the monitoring process will be badly affected.

2.4.3.12 Lack of mechanisms to support adding new MIBs dynamically to a system

MIBs are continually being upgraded to reflect the new capabilities of network devices. The procedure to upgrade a MIB in the manager console is the static compilation of MIB text files. The output of the compilation process is a set of objects which will be used by the manager to interface to agents with the new MIB. [5] In the case of the agents there is not the adequate support to upgrade the MIB dynamically. [34] A standard way to dynamically add support for new MIB modules to a system is needed to allow SNMP to adapt itself to the continuous change in the network environment. [35]

2.4.3.13 Lack of scalability to manage a bigger system

The traditional model in SNMP of having a central single management station in charge of many agents runs into scalability problems when the network grows too large. [36,37,38,39] This is because the amount of data that the management station must retrieve and process grows exponentially as the size of the network increases. [23] It means that a centralized approach, like SNMP for network management is not the most adequate solution for the efficient management of large computer networks. [40] The clear solution is to delegate tasks from a high level manager to subordinate managers or in another words the management by delegation. [30] The use of delegation-based management can significantly improve the monitoring capabilities of management applications. [21,41,42] The IETF has become aware of the problem and it has set a Distributed Management group to advance the progress of the Manager-to-Manager MIB. [43]

2.4.3.14 Lack of flexibility to update the system

Network management applications typically need to extract useful information from the MIB embedded into the SNMP agents and perform computations on this information locally. [44] In a growing number of scenarios this is either impossible or impractical. A typical scenario where an alternative proactive approach can be better than the current one is where quality of service must be provided. [45] The SNMP framework does not have the ability to switch from a passive management system to an active management system where the agents have the enough intelligence to do actions by themselves. [46] From the outset, it was assumed that agents in SNMP would be simple devices that could not support anything more than the simplest protocol functionality. [30,19,47,48] This is untrue today. Today's network devices often contain processors and memory exceeding that of our management platforms of a few years ago. This infrastructure provides the basis for making the devices capable of, to some extent, managing themselves. [19,45] There are several examples which illustrate the benefits of more intelligent agents:

- The ability of the MIB to make assertions. An assertion would be a Boolean value computed from the existence of certain conditions. This will facilitate a quicker response to a problem.

- The creation of dynamic MIB variables based on real MIB values. This feature could capture new empirical knowledge which today is rarely recorded. [46] The function can be as simple as an error counting gauge variable divided by SysUpTime to give you an average error rate, or more complicated like the

correlation of large number of related events, which are masking a single problem. [49]

- The delegation of functions from the main manager to the agents can create a semi-autonomous system. [45] This new environment is possible with the introduction of the needed infrastructure in a device that permits the selection and upgrade of the functions to develop.

- The creation of new elements like a roaming manager which can substitute the current system of polling with a new system where every agent, temporarily, will host a pseudo manager who will collect relevant information and instruct the agent to behave accordingly and eventually report back to the real manager.

- The use of a proactive network management approach will permit the recognition of problems, as they are occurring or about to occur and to take measures to prevent them without a network administrator's intervention. [50] It could, for example, provide a solution to the problem of preventing network outages and congestion that the typical network management tools cannot manage.

- The acquisition of an atomic SNMP table view. Also support for the selective retrieval of objects that meet a specific selection criterion. [20]

2.5 SUMMARY

This Chapter has provided a briefly summary of network management technology. The analysis largely related to SNMP because it is considered to be the de facto network management standard. Its popularity is based on its simplicity and while its design was

adequate when first defined, it has serious weaknesses when applied to current network environments.

One of the major contributions of this Chapter is the detailed analysis of the weaknesses of SNMP. [104]

Chapter 3

ACTIVE NETWORKS

3.1 INTRODUCTION

This Chapter presents Active Networks. Section 3.2 introduces the concepts of Active Networks and they are compared to the more 'passive' traditional communication networks which merely switch packets. In contrast Active Networks carry out code and data in the packets and the Active Nodes execute the code which processes the data.

A review of the major Active Networks technologies is given in Section 3.3. This is followed by a comparison of these technologies in Section 3.4. It is argued that many of the Active Network technologies operate in more or less the same way although some have been designed for specific application domains. ANTS was chosen as the technology for this research. It is open source and the most widely used of all the technologies. More importantly, it is the only technology where the Active Network services are written in Java. This is an important criterion for this research as the test-bed will be Java based.

For this research a number of SNMP and more general network management standard problems will be solved. Section 3.5 discusses why particular problems were selected.

3.2 INTRODUCTION TO ACTIVE NETWORKS

An Active Network is a virtual infrastructure overlay on the real network which provides the elements that facilitate the evolution of network services[3]. Figure 1 shows a typical Active Network topology with active and traditional non-Active Nodes. The architectural design of Active Networks can be divided into two main categories, the Discrete approach and the Integrated approach. [2] Both use special (active) network packets to deploy the service code in the Active Nodes inside the network. A requested service is executed by requiring the active packet to carry a service program identification which makes each service automatically and uniquely recognisable among other services. The Discrete approach preloads the service before the service execution can begin. Examples of Active Networks technologies which use this approach are the ANTS architecture proposed by MIT (Massachusetts Institute of Technology) [10], the CANES architecture proposed by Georgia Institute of Technology [51], and the DAN architecture proposed by Washington University and ETH Zurich. [52] Thus, once a new service is deployed Active Networks packets carry only the user data which will be processed by the service.

[3] Through the thesis the term service means a Active Network Service unless specified otherwise.

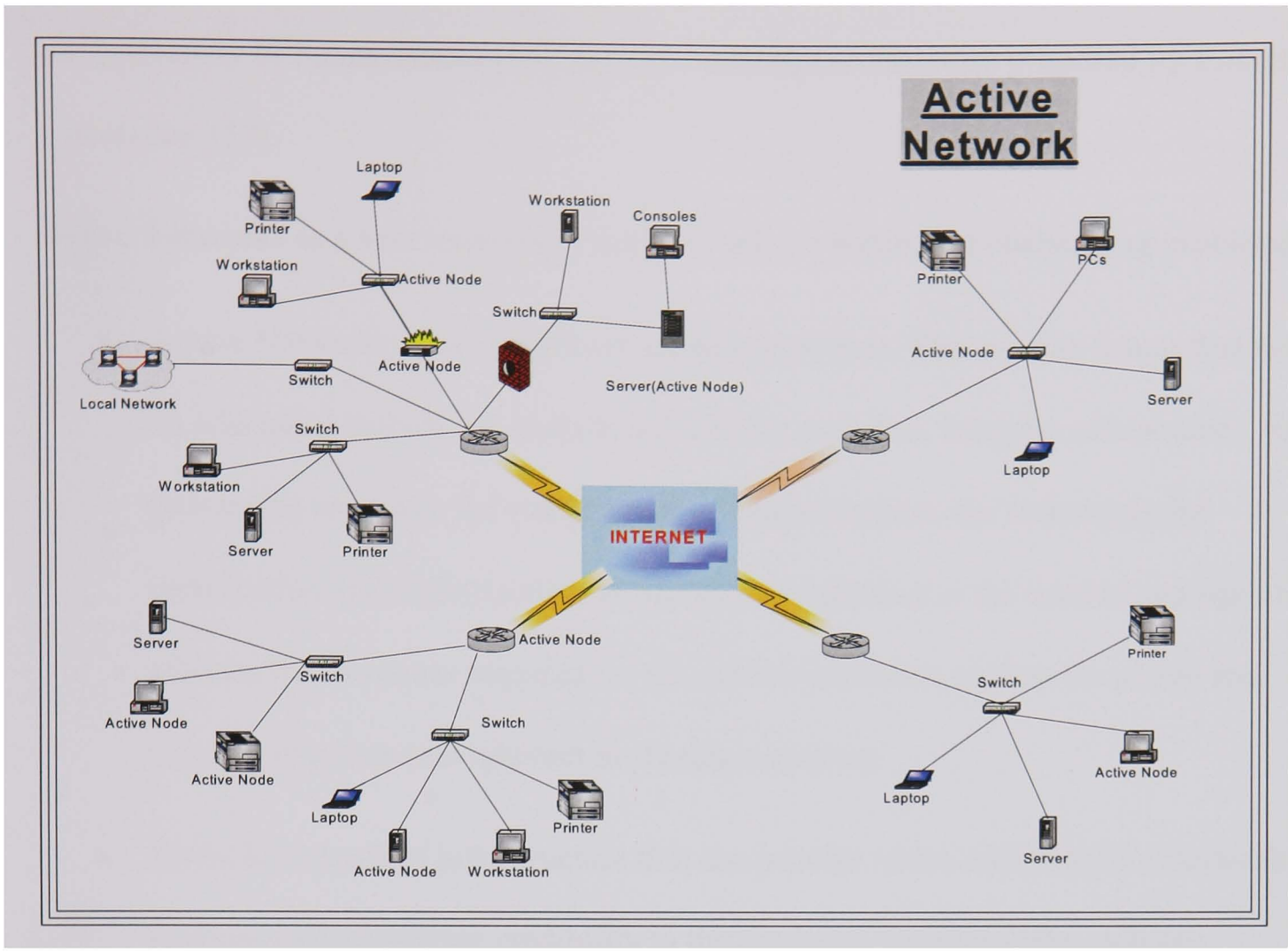


Figure 1. A typical Active Networks Topology

The Integrated approach to Active Networks does not need to have the services preloaded in the network; instead it requires that the active packets carry both user data and the service program. Examples of this architecture are the Active IP proposal of the MIT group [53], the Smart Packets project at BBN Technologies [54] and the M0 architecture proposed by the University of California at Berkeley and the University of Zurich. [55] Some Active Networks architectures permit the use of both approaches. Clearly the Integrated approach is only viable when the service code is small, whereas the Discrete approach limits the flexibility as service code must be preloaded before it can be invoked. Examples of the mixed approach (Discrete and Integrated) are the SwitchWare project at

the University of Pennsylvania [56], and the NetScript architecture proposed by Columbia University. [57]

Active Networks as a new technology needs to solve a number of challenging problems:

- Active Networks raise significant security problems. The security issues that need to be addressed include the security policies for switches, integrity of resources, the processing resources for certification and authorization, the techniques for certification and authorization. A further consideration is the need to appropriately allocate the resources required for the service execution against the safety and security requirements inherent in sharing resources.

- To build a standard infrastructure that can provide value-added Active Networks services to demonstrate credibility to the academic world and the industry.

Active Networks offer a new networking paradigm and apart from the problems noted above, the ability to download new services into the network infrastructure will lead to a user-driven innovation process in which the availability of new services will be dependent on their acceptance in the marketplace. This is similar to what is happening in other areas of the Internet, for instance search engines. Active Networks present an opportunity to change the structure of the networking industry, from a 'mainframe' mind-set, in which hardware and software are bundled together, to a 'virtualized' approach in which hardware and software innovation are decoupled. This approach allows new services to be deployed at a faster pace and that can be sustained by vendor driven consensus and standardization activities.

Currently, the most important Active Networks research projects are concerned with congestion control [58], multicasting [59,60,61,62], web caching [63,64], and its application to network management. [65,66,67]. The focus of this research is the application of Active Networks to solve a number of key network management problems identified in Chapter 2.

It was explained in Chapter 2 that network management systems typically operate by having the manager periodically poll the agents for the values of MIB variables, and check for anomalies in the agent responses. This approach to monitoring the network has served well in the past. Nevertheless, due to the increase in the number of nodes and the complexity of the network, this approach is no longer adequate. Managers are overwhelmed with a large amount of information, very often redundant, as the packets that arrive may simply report that there was no change in the state of the agent. However, in the event that a problem exists, the round-trip time that has elapsed before the manager receives the agent response and the subsequent reply to the agent can be significant, and in some instances the action requested by the manager is no longer valid. In contrast, current network management systems are primarily designed for network devices that have high availability (are seldom 'down') and have enough local resources to host reasonably more intelligent agents. Thus, it is sensible that the approach to network management is changed so that the agents can become self-managing and thus reduce the load on the manager. In this way network management may scale as the network grows.

Active Networks could be the natural solution to the network management scenario, and the network management problems identified in Chapter 2. For instance, by making the internal nodes of the network active it is possible to distribute the network management functionality. This one change will decrease the delay in the manager detecting and reacting to a fault and it will also reduce the bandwidth used for network management purposes. Another possibility is to deploy services to the agents, where agents are now Active Nodes. These agents can now act autonomously detect and react to their own faults without any supervision from the manager. Other types of services can also be deployed. Active Networks have other advantages, as they also support naturally asynchronous operations and are portable across operating systems.

To summarize, using Active Networks for network management gives the following advantages:

- Problems are tracked quickly or are reported automatically without the need for polling.
- Network management can be distributed in the network. This reduces the response delay and network resources requirements.
- It is possible to put the intelligence in the agents; this reduces the load on the manager and also reduces network resources requirements.
- The exchange of network management information is minimised and tailored precisely to the need of the manager.
- A change of policy is easily achieved by simply downloading a new service in the agent.

- The communication between manager and agent is asynchronous.
- Active Networks architecture solves the portability problems caused by the use of different operating system in the network devices.

3.3 MAJOR ACTIVE NETWORKS PROJECTS

The main goal of this research is to investigate the possible use of Active Networks technology to solve the current network management problems. This section reviews the major Active Networks technologies available in order to decide which is the most suitable for solving network management problems identified in Chapter 2.

3.3.1 ACCELERATING NETWORK EVOLUTION WITH A SOFTWARE SWITCH FOR ACTIVE NETWORKS. [56]

3.3.1.1 Introduction

This project is a joint project between the University of Pennsylvania and Bellcore Communications and aims to develop a set of software technologies ('SwitchWare') which will enable network behaviour to be modified. The main objective is to balance the flexibility of a programmable network against the safety and security requirements needed in a shared network. The software is distributed and runs in a robust programming environment or switch. The key objective is to make the basic network service programmable on per user (or even per packet) basis as other Active Networks Project. The objective is to develop a programmable switch approach that allows digitally signed, type-checked services to be loaded into the nodes of a network.

The SwitchWare Active Networks architecture comprises three layers:

- Active packets: these are mobile programs that replace traditional packets.
- Active extensions: these are static elements which provide services to SwitchWare services.
- Secure active router: this forms a high integrity base upon which the security of the other layers depend. Integrity checking and cryptography based authentication, along verification techniques copied from programming languages, such as strong type checking, are the basis for security.

3.3.1.2 Active Packets

Active packets replace the traditional network packet with a mobile program, consisting of both code and data. The code part of an active packet provides the control function of a traditional packet header, but it can interact with the environment of the router in a more complex and customizable way than the simple table lookup provided by conventional packet headers. Similarly, the data in the active packet program replaces the payload of a traditional packet, but provides a customizable structure that can be used by the program. Basic data transport can be implemented with code that takes the destination address part of its data, looks up the next hop in a routing table, and then forwards the entire packet to the next hop. At the destination, the code delivers the payload part of the data. The active packet is encapsulated in a standard UDP packet.

A new language called PLAN (Programming Language for Active Networks) is written specifically to program active packets.

Its execution model includes a mechanism for remotely evaluating PLAN programs on other routers; these mechanisms are the means by which PLAN programs transport themselves through the network. PLAN programs are strongly typed to provide safety, and can be statically type-checked before being injected into the network. A PLAN program cannot by itself manipulate node-resident state. To compensate for these limitations, PLAN programs can call node-resident service routines, which can authenticate or use other more heavyweight mechanisms to provide security on a demand basis. Internally, a PLAN program consists of code, plus an entry point into that code, plus any data that make up the arguments to that initial function. Furthermore, PLAN provides a number of mechanisms for limiting the resources used by an active packet: Each PLAN packet has a resource bound, much like IP's Time-To-Live (TTL) field, which serves to bind the total number of hops a packet, and any packets it creates, can take. In contrast, every packet has limited CPU and memory allocation on the routers. SwitchWare has developed a test bed called PLANet, based on the current Internet to test PLAN programs.

3.3.1.3 Active Extensions

In SwitchWare active packets are deliberately limited in power, making it impossible for them to be used to implement arbitrary protocols or functionality. Node-resident extensions form the SwitchWare architecture middle layer which combined with the active packets creates a service. As an example of this complementary relationship, Active packets can

call extensions to provide authenticated services, making it possible to avoid default authentication for active packets. Active extensions can be dynamically loaded or be part of the base functionality of the router. They are not mobile: to communicate with other routers they use active packets. This is the main difference between active packets and extensions: Active extensions must be executed entirely on a particular node. They can be written in general purpose programming languages (in several prototypes SwitchWare uses the language Caml) and can use a variety of security mechanisms, including cryptography-based authentication and program verification. Their greater flexibility provides an important trade off with active packets since complex protocols and systems are implemented in SwitchWare as a mixture of PLAN and router extensions.

In general, SwitchWare expects that active extensions will be loaded into routers, and will then provide services to many active packets. Thus, it is reasonable to subject active extensions to heavier-weight security checks than active packets. Because of this heavier-weight checking, active extensions can be allowed access to facilities in the router that active packets cannot. In particular, creating or changing state in a router, or direct access to the routers network interfaces, must be done by extensions.

3.3.1.4 Secure Active Router

In SwitchWare a secure active router infrastructure is called Secure Active Network Environment (SANE) and forms the lowest architecture layer. While the top two layers emphasize support for several forms of dynamic update, the lowest layer is primarily static. The goal of this layer is to guarantee integrity to the other two layers built, incurring

minimal costs while the system is in operational state and to maximize system security under a minimal set of assumptions about trusted components. SANE establishes a minimal set of system elements (e.g. BIOS, cryptographic material, ..) upon which system integrity is dependent. It then builds an integrity chain with cryptographic hashes on the image of each succeeding layer in the system before passing control to that image. It also provides a public-key infrastructure that can be used for cryptographic authentication of module sources.

SwitchWare imposes a number of policies in which a complex set of checks involving signatures and type verifications is carried out. In particular, any code that is downloaded into SwitchWare routers should be strongly typed, since this provides some basic guarantees that the router's integrity will not be compromised by the code. SwitchWare applies heavyweight security checks on active extensions, which may represent major releases of Switch code, and more lightweight security checks on active packets. This approach allows the network architect to balance security concerns against performance requirements. The security model of SwitchWare considers public, authenticated and verified facilities.

3.3.2 NETSCRIPT. A LANGUAGE-BASED ACTIVE NETWORKS ARCHITECTURE. [57]

3.3.2.1 Introduction

This project developed at Columbia University aims to provide a language system to construct, deploy dynamically and execute active-element programs in network nodes. The

NetScript project takes a functional language-based approach to capture network programmability. NetScript is a strongly typed language that creates universal abstractions for programming network node functions. Unlike other Active Networks projects that take a language-based approach NetScript is being developed to support Virtual Active Networks as a programmable abstraction. A distinguishing feature of NetScript is that it seeks to provide a universal language for Active Networks in a manner that is analogous to Postscript. Just as Postscript captures the programmability of printer engines, NetScript captures the programmability of network node functions. NetScript communication abstractions include collections of nodes and virtual links that constitute virtual Active Networks.

3.3.2.2 NetScript component model

The NetScript component model is based on well-known object-oriented programming concepts of interfaces, components, attributes, and globally unique identifiers. In NetScript, an interface is a group of semantically related methods or functions. A component is an implementation of the interface and can implement one or more interfaces. Each interface and component is identified by a globally unique identifier (GUID). Components in NetScript could be Java Beans or Java-wrapped native (e.g., ActiveX) components. Interfaces and components are advertised in a distributed catalogue based in LDAP.

The NetScript scripting language is an extension of the BASIC programming language. In addition to standard control constructs, the language has a few NetScript specific additions to create component instances and to make method invocations. For security reasons the

language has no vocabulary for system operations such as direct memory access, file access, and network access. In NetScript the programming model is interface centric. When writing scripts, the system manager simply chooses some functionality (syntactically and semantically characterized by an interface) from a catalogue and the runtime locates that functionality in the form of a component. The script first attempts to create a component that implements the interface wanted, then, the runtime uses the component catalogue to locate the actual host for a component that implements the interface. When such a host is located, the script execution is suspended until the missing component is obtained. The component implementing the interface can be written in pure Java or other language, as long the component supports the interface methods and semantics. An example of component functionality is the database access operations, read, write, etc.

3.3.2.3 NetScript runtime system

The NetScript runtime has been implemented in Java. The runtime has a script interpreter, an execution engine, and a set of shared services such as directory, instance management, and communication. The run-time also has built-in support for security and access control, garbage collection, monitoring, and failure detection and recovery. The NetScript runtime environment has command-line and web-based tools for the user to launch, monitor, and control executing scripts. To start the system a script is first launched on a machine using one of the NetScript tools. The NetScript runtime initializes some data structures and starts executing the script. Every runtime has a per-script instance manager that keeps track of component instances that the script may have created. On method invocation, the runtime

queries the instance manager to decide if the script needs to access other component instances.

The component entry in the catalogue specifies its location and the name of the jar file in which the component is packaged. The NetScript runtime environment finds component location information from the catalogue, unpacks the jar file, and instantiates the component. On the fly, the runtime contacts the component catalogue to locate a component that implements the required interface and then either migrates the script to the component location or downloads the component to the script's location, instantiates the component, and stores the handle for the instance in a script variable.

The classes and native libraries such as Windows or UNIX libraries are packaged as jar files for distribution.

3.3.2.4 NetScript security system

NetScript provides security mechanisms for attaching a ''principal'' with a script. Upon receipt of a script, the NetScript runtime authenticates the principal and verifies that the principal has execution rights on the local host. When a script accesses a component, the principal attached to the script is passed over to the component catalogue. Component entries in the catalogue list principals that are allowed to use the component. A component location is returned to the requesting runtime only if the provided principal is included in the list. The runtime uses per-script instance managers and class loaders to implement isolation. The NetScript environment also does not allow for editing a running script because of possible security breaches, however, it provides support for locating a script,

retracting a script from any location, or killing a script. Scripts are identified by GUIDs generated and assigned to a script at the start of its execution. Any errors that are encountered by a script, including exceptions generated by components, are reported to the user when the user requests for the status of a script that has failed.

3.3.3 THE ACTIVE NETWORKS TRANSPORT SYSTEM TOOLKIT (ANTS)

The Active Networks Transport System (ANTS) is the most widely known Active Networks platform. It was developed at MIT by David J. Wetherall. [11] The main objective of ANTS is to establish a platform, flexible enough to allow the creation of new networking services without the long process of standardization. A Service, as defined by Wetherall, is composed of a number of ANTS standard elements: An Application, a Protocol, a number of Capsules and one or more Extensions.

In every Active Node one or more services can be hosted. A service is another name for what would traditionally be called a user application (this is not the same as an application element in ANTS). Execution of a service is initiated by an application element. The application element requires one or more capsules. A protocol element is used to define the service structure. A capsule is a special type of network packet used by an application element to communicate with others Active Nodes. It is designed to carry both data and code. Depending on the service complexity, its functionality could be implemented by capsule elements but it is more commonly implemented using extensions with the capsules used to transport information between nodes. An extension is a service element used to extend the functionality of an Active Node. Every Active Node is connected to its

neighbours with channels; channels are used by ANTS for the network communications. Figure 2 shows a diagram of the ANTS Active Networks architecture with all the elements present.

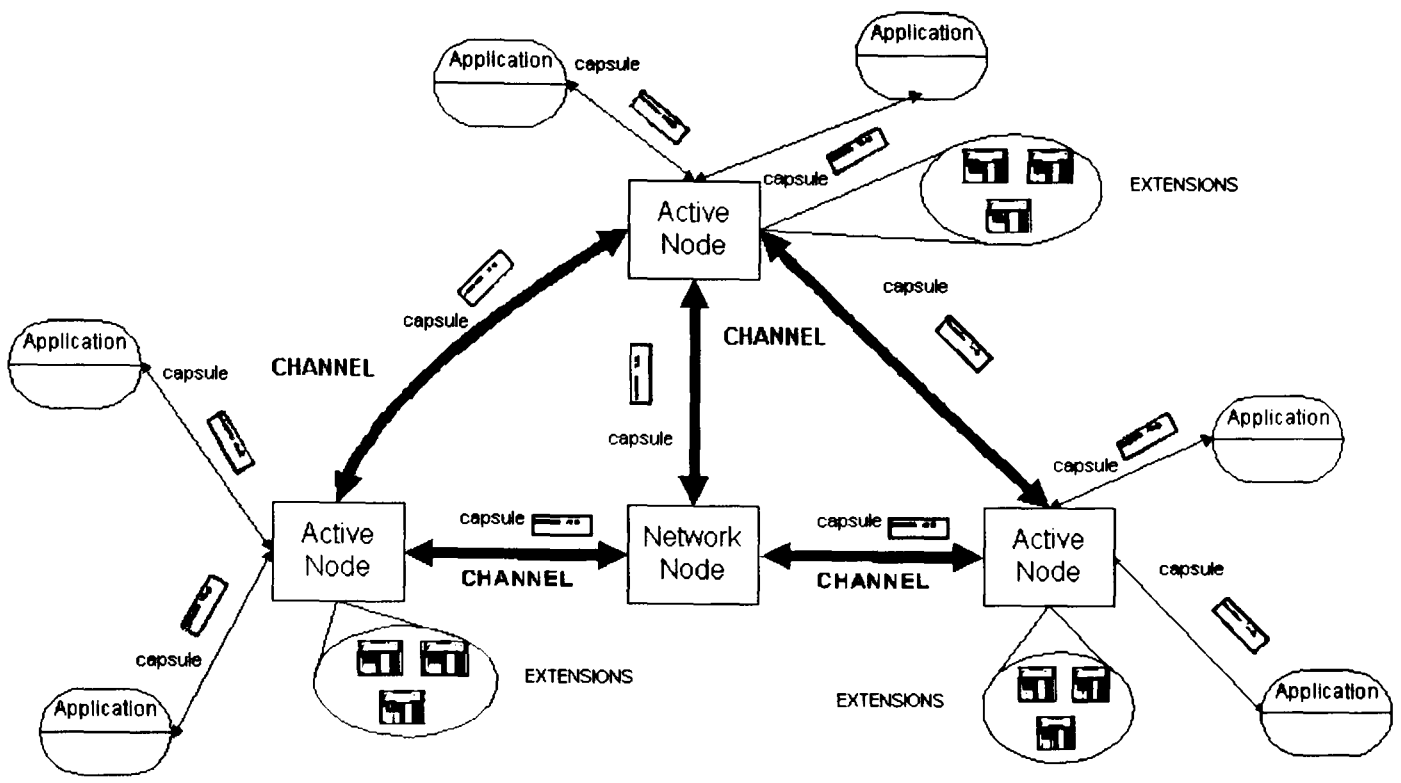


Figure 2. ANTS Active Networks architecture.

It was mentioned earlier that a mix of active and traditional non Active Nodes is the most common network topology in an Active Network. This illustrates the flexibility that Active Networks provide. Before a user can make use of an Active Network service the Active Network layer (software) must be overlaid on the existing communications architecture. The design of ANTS permits the incremental deployment of Active Nodes in a network; this avoids any disruption of network service commonly experienced when a new technology is introduced.

The architecture of ANTS is described by three basic concepts:

- The Programming Model, which is used for the creation of new services.
- The Demand Pull Code System, which permits the dynamic distribution of the capsules.
- The Node Operating System.

The next three sections will explain, in detail, these three concepts.

3.3.3.1 The Programming Model

The Programming Model utilises the ANTS elements to create new services, mainly Extensions and Capsules. It also uses the API to access crucial system services provide by ANTS.

3.3.3.1.1 Extension Element

An Extension is the element used to extend the functionality of the Active Node. This is therefore a critical element for the developer. ANTS provides an abstract class called Extension which the developer then extends to include the appropriate functionality for their service. The use of extensions in a service is not compulsory but, except when the services have trivial functionality, extensions are the elements that contain most of the code. Every extension is implemented as a separate thread. The Programming Model provides a procedure for the static installation of extensions in a node. As part of this research a new procedure to dynamically install extensions has been designed and

implemented. A more detailed explanation of this mechanism could be found in the Chapter 5 where the SNMP Active Networks Toolkit is presented.

In summary, while the extension element is a critical element in service provision, because the service code is written by the developer, ANTS only needs to provide the abstract class. Thus there is little to explain.

3.3.3.1.2 Capsule Element

A Capsule is a special type of packet, which is used to transport information between Active Nodes. The main difference between a capsule and a traditional network packet is that a capsule carries data and code. In addition capsules are used to initiate the execution of the service.

Capsule format

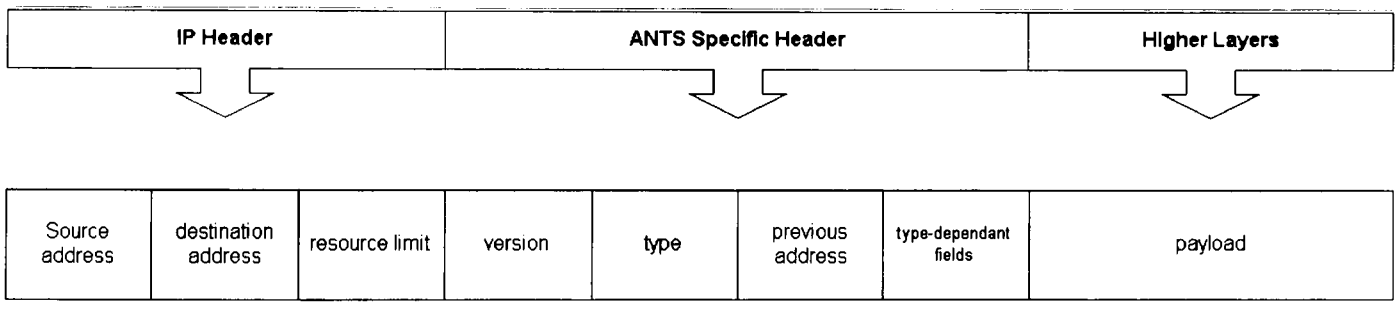


Figure 3. ANTS Capsule Format.

Every capsule has a number of fields. Figure 3 shows the format of a capsule. From left to right these fields are:

- The origin and destination address of the packet.
- A resource limit field with the same functionality that the IP Time to Live field
- A number that identifies the ANTS toolkit version.

- A type field which stores a fingerprint that identifies the capsule in the global domain. e) The IP address of the last node visited by the capsule.
- A number of fields set by a service in the capsule. Because the service could be distributed over a number of Active Nodes, these fields are required to communicate information between the Active Nodes participating in the service.
- A payload which is the service code to be transferred.

All of these fields perform important functions in the operation of a service, but it is the fingerprint that differentiates capsules associated with different services. Thus the fingerprint is the unique service identifier. The fingerprint design is based on two important properties: uniqueness and simple check of equality. The technique to get these properties is by the known security algorithm, MD5. [80]

A number of steps are required to create a fingerprint.

- The service capsule fingerprints are created by using the bytecode of the capsule as the input parameter to the MD5 algorithm.
- Because a service may comprise a number of capsules which may be associated with different protocol elements. Capsules fingerprints related to the same protocol within the same service are used as input parameters to the MD5 algorithm to compute the code group fingerprints. This fingerprint uniquely identifies the code group among the others code groups in the service. This importance of the code group fingerprints will be explained in the next step.

- All the code group fingerprints associated with a service are used as input to the MD5 algorithm to compute a service fingerprint.
- Finally, a combined fingerprint value is computed by the use as input parameters the capsule fingerprints, the code group fingerprints and the service fingerprint. This combined fingerprint value is stored in the Type field in the capsule. Figure 4 illustrates the process followed.

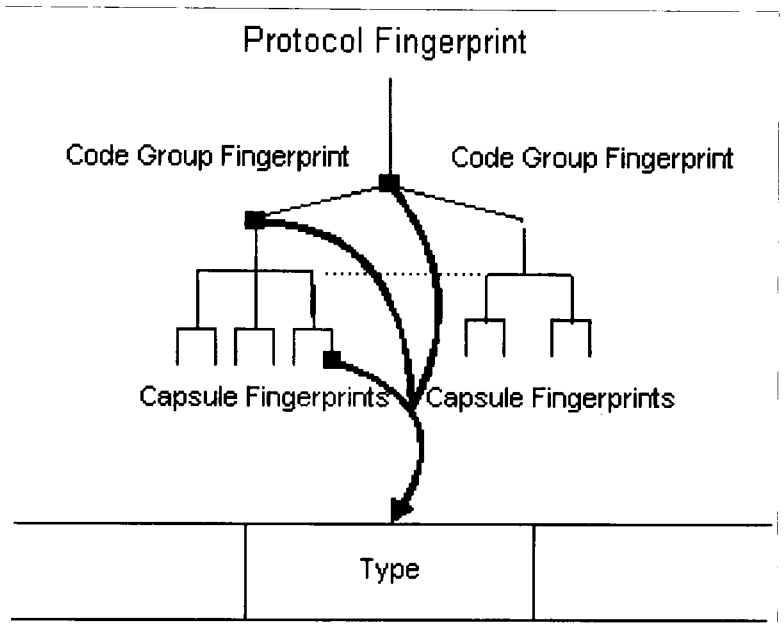


Figure 4. Capsule Fingerprint structure.

3.3.3.1.3 Capsule processing model

The capsule processing model uses the combined fingerprint value for capsule processing. The process starts when a capsule arrives at a node. If the node is an Active Node, the capsule type field (which stores the fingerprint) is checked to verify its authenticity against the capsule code; otherwise the capsule is treated as a normal IP packet. If all the capsule code is not already in the Active Node the Demand Pull Code System is activated (the Demand Pull Code system will be described later in the Chapter). The authentication

process recomputes the combined fingerprint; this is similar to the CRC generation and checking process used in communication networks. If the authentication process fails then the capsule is discarded, otherwise the code in the capsule is executed. The execution of the capsule is the beginning of the service execution.

The processing of the service happens inside an execution 'sandbox' which is characterized in a number of ways:

- The execution of a capsule is only permitted for a limited period of time. The Active Node operating system implements this control by using a capsule timer. If the time limit is exceeded, the node aborts the execution of the capsule code, and then subsequently destroys the capsule code.
- Only a limited amount of information is explicitly transported in the capsule. This information is associated with the service and thus can only be interpreted by the service itself.
- If the capsule code contains external references (invocation to an extension) then these must be already stored locally. This implies that there are no delays in the execution of the capsule when the code is available.
- It is always possible to make calls to functions provided by the Active Node operating system API. Also a capsule can invoke other capsules of the same service. For this call to be permitted, the code of both capsules must be available locally.

Finally, a number of errors can rise when a capsule is processed: lack of resources in a node for the execution of the capsule, default path not set, etc. To deal with these errors in an efficient and robust manner, the errors are classified locally as exceptions.

3.3.3.1.4 ANTS API

A service developer makes use of the ANTS API to build capsules and extensions. The methods provided by the API can be divided in three main categories:

- Methods related with the node environment. This category groups the methods which allow information about the Active Node to be retrieved. Examples of this category are, the method GetAddress, which retrieves the node IP address and the method Time that returns the local time.

- Methods related to the node's storage capacity. These methods manage the Active Node's temporal cache. The information stored in the cache can be used to facilitate communication between capsules. Examples of these methods are Get, Put and Remove, etc.

- Methods related to the control of capsule processing. The service developer uses these methods to decide whether a capsule should be forwarded to the next node or delivered to the application element. The method RouteForNode facilitates the transfer of the capsule to the destination node via the default path. Resources are decremented in every Active Node visited. The method DeliverToApp delivers the capsule contents to the application element which has requested the service.

3.3.3.2 The Demand Pull Code Distribution System

Once a service has been defined in terms of capsules and extensions, it must be deployed in the network before it can be used; this is the responsibility of the Demand Pull Code Distribution System.

The objectives of the Demand Pull Code Distribution System are:

- Adapt to unforeseen changes in the path in the event of, for instance, a node failure.
- Scale as the size of the network increases.
- Minimize the size of stored code and the distance over which it is transferred.
- Minimize the elapsed time since a capsule arrived at a node until the necessary code is received.
- Prevent code spoofing and denial-of-service attacks.
- Prevent congestion in the network.

To achieve these objectives the technique used by ANTS is to pull code along the capsule path and to store the code in the Active Nodes while the service is in use. Note, the Demand Pull Code Distribution System deals exclusively with distributing capsules. ANTS does not provide a similar distribution system for extensions. A detailed technical explanation of how the Demand Pull Code Distribution System works is given in the sections following.

3.3.3.2.1 Initialization of the distribution system

Before a service can be run, all its code must be stored in the access Active Node, because the code distribution system begins here. Code in this context means all the capsules, the necessary protocols to organize them and the extensions needed. ANTS defines an abstract Application class. This class must be extended by the service developer and it fulfils the same role as the Main method in a Java application; it is used to start the service. Once this application class is started in the access Active Node, it registers the service code with the access Active Node. The Active Node can now create the fingerprints for the capsules and the necessary code to begin the bootstrap deployment of the service. The application class can now send service capsules to the next node in the path using the network topology information available. The Demand Pull Code Distributed System will be triggered when the first service capsule arrives at the next Active Node.

3.3.3.2.2 Demand Pull Code Protocol

In the ANTS Active Networks nodes, the automatic system of code distribution is invoked in response to the arrival of a capsule requesting code that is not currently stored the node's cache. In particular, the Demand Pull Code Distribution System is used to request the necessary code from the upstream node from where the capsule arrived. When the code arrives it will be stored. The demand pull protocol employs an unreliable transfer protocol. The four stages of the demand pull protocol are shown in Figure 5. Note that the numbers shown in Figure 5 correspond to the stage numbers explained below.

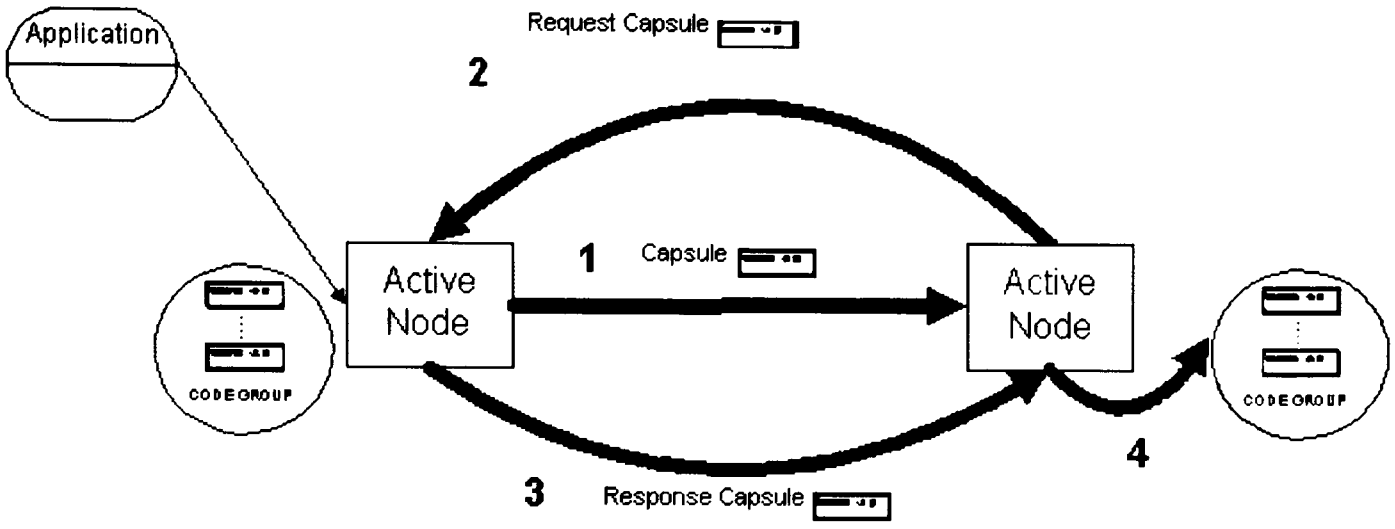


Figure 5. Capsule Fingerprint structure.

1. A capsule is sent towards the next Active Node with the current Active Node address set in the Previous Address field.

2. When the capsule arrives at the Active Node, the Node Operating System discovers that other capsules, needed for the service, are missing. Using the address in the Previous Address field, the Node Operating System then, sends a Code Load capsule requesting the missing capsules to be transferred.

3. The Code Load capsule arrives at the previous node, which responds by sending the missing capsules.

4. Once the code transfer has finished the Active Node Operating System can resume the execution of the capsule that initiated the service.

The capsule fingerprint is used as the key to store the capsule in the cache. Each cache entry maintains a set of capsule information: the capsule code, a field that identifies the parts that must be loaded for execution and a timestamp. The cache is organized as a least-recently-used cache. When a capsule arrives at the Active Node, the cache is searched. If an

entry exists for the capsule then it is executed immediately, otherwise the process of code loading is initiated:

3.3.3.3 The Active Node Operating System

The ANTS operating system has two major responsibilities, resource management and to ensure the integrity of service code.

Resource management

The resources of interest to the operating system are network bandwidth, cache and processing time. The overall resource management policy is to avoid starvation rather than achieve fairness.

The bandwidth allocated to a capsule is defined by using the Resource Limit field in the capsule. This field is used in a similar way to the Time to Live field (TTL) in IP. Only the operating system is allowed to update the Resource Limit field. When the Resource Limit Field value is zero, the capsule code is removed from the cache. A timer with a fixed value is used to limit the amount of processing time allocated to a capsule.

Protection Mechanism

The protection mechanism ensures the integrity of services. In particular, it ensures that;-

- The service code cannot maliciously or inadvertently corrupt the operating system.
- The code from one service cannot interfere with code from another service.
- The code from the service cannot be corrupted

- The code cannot consume an excessive amount of resources.

This is achieved by the separation of the code and state of the different services.

3.3.4 CANEs: Composable Active Networks Elements.[51]

This project, developed at the Georgia Institute of Technology, aims to design and implement a set of enhanced Active Networks services that support the installation of new, user-defined capabilities as well as customization of basic services. In the architecture proposed, applications invoke enhanced services through a generic interface that supports multiple forms of function specialization. Examples of these services includes congestion control, network caching and media transformation. The interface of the services is based on a simple packet processing model and is not tied to any particular language. The architecture supports varying degrees of network service customization, from parameter selection to setup of completely user-defined services. The mechanism developed is compositional from primitive elements, allowing the properties of the composite service to be derived from the properties of the elements. A composition method is specified as a programming language with enhanced language capabilities that operates on components to construct programmable network services. The CANEs project aims to define and to apply service composition rules as a general model for network programmability.

The project is putting its efforts into solving network congestion problems by allowing applications to request that specific node algorithms (e.g., lossless compression, selective discard, and trans-coding) be invoked during periods of congestion.

3.3.5 CONVERSANT - AN ENVIRONMENT FOR REAL-TIME, SECURE ACTIVE NETWORKS.[78]

This project, developed by The Open Group Research Institute of the Open Software Foundation, Inc aims to deliver Active Networks support for command and control applications by creating a networking environment and infrastructure that extends the Active Networks vision. Specifically its main goal is to provide an Active Networks node designed to defend against denial-of-service attacks. The system proposed supports: a) dynamic deployment of specialized protocols to support adaptability. b) The coexistence and interoperability for custom protocols with standard protocols. c) A common protocol environment that allows the same protocol implementations to be deployed in network nodes, endpoint systems or applications. d) Secure use of Active Networks and resource management. Conversant uses Java as its execution environment and vehicle for mobile code. Most of Conversant's efforts were devoted to controlling the CPU and memory resources by modifying the standard Java Virtual Machine.

3.4 COMPARISON BETWEEN MAIN ACTIVE NETWORKS PROJECTS

In general, the three major Active Networks technologies presented (the others are created to deal with specific problems) has a similar architecture design pattern. All of them have special network packets used to trigger a Service in an Active Node. The functionality of the Active Node can be extended using Extensions and the Active Node operating system provides the Service running environment.

ANTS differs in some aspects from SwitchWare. In ANTS a capsule usually does not carry the complete Service code, but instead partial code and a unique Service identifier used to download the code missing in an Active Node. In contrast, SwitchWare uses a hybrid approach based on 'active packets and nodes', i.e., small services have the code already embedded in the Plan packet while Active Nodes can provide any complex code that is dynamically downloaded if it is needed for more complex services. It means that for small services a PLAN packet is larger compared to an ANTS capsule. The transport of active code in every packet is justified for specific services; in other instances it could be a waste of resources. In contrast, considering the ANTS loading process, a highly changing network topology could led to several code downloads that can overload network links and consequently reduce global network performance. It can be said that the 'out of band approach' of ANTS out performs the 'in-band approach' of SwitchWare in minimizing the amount of control data to be carried by the network for long lived communication sessions and services, while short live sessions are better suited to SwitchWare.

NetScript takes a functional language-based approach to network programmability, treating the network as a single programmable abstraction of Virtual Network Engines (VNE) interconnected by Virtual Links (VL). The installation of software to be installed at the intermediate node in this Virtual Network is performed through agents that are transmitted from the end nodes to the intermediate nodes. Specifically, the first packet contains the information needed by the VNE to process the packets that arrive subsequently belonging to the same stream. The subsequent packets that arrive at the VNE are identified by the stream id to which they belong to and processed following the instructions of the first

packet. Again this approach can be suitable for small services, where the first packet instructions are simple, but not suitable for more complex services.

Both SwitchWare and NetScript use their own created language to develop Active Networks Services, meanwhile ANTS uses Java for the same purpose. The use of one specific language to create Services can improve the security but it could limit the Service creation elements to the developer.

From the analysis done to the main Active Networks projects it is possible to conclude that they are not so many differences between them. An important one is the language used to create Active Networks services. In the case of this research, the election of the language is quite important due to the need to build a real network management system.

The next Chapter will present the Java Management Extension (JMX) specification which is a Java standard technology to build a network management system. The fact that JMX is a Java standard makes it natural to choose ANTS as the Active Networks project to use and this is the main reason to select it as the Active Networks technology where the Network Management Active Networks services will be created. Besides, ANTS is open source and it means that it could be possible to create solutions to any ANTS code bugs, without waiting for bug code updates. Finally, ANTS is one of the most representative Active Networks projects as the multiple references reveal [69,70,71,72,73,74,75,76] and it is the base for a number of different research projects. [77,78,79]

3.5 SELECTED ACTIVE NETWORKS SERVICES

The last Chapter discussed Network Management issues and the limitations of SNMP. As part of this research, a critical investigation into the strengths and weaknesses of SNMP was provided. The purpose of this research is to answer the question, 'Can Active Networks solve all the documented problems of Network Management Systems?'

To obtain an answer to this question a number of SNMP problems will be selected and a solution, in the form of an Active Networks service, will be proposed.

The problems of SNMP can be viewed as specific inefficiency problems related to the way in which SNMP implements particular services (commands) and more fundamental operational problems, shared by other network management standards.

The specific SNMP inefficiency problems chosen are GetBulk and Threshold. GetBulk is the command used by SNMP to retrieve tables and the way in which this operates was described in Chapter2. Typically SNMP requires many messages to retrieve tables when sensibly one 1 message should be required. Thus this is an obvious candidate to compare the standard SNMP command with the equivalent Active Networks service. Similar inefficiencies exist with the Threshold service.

The more fundamental operational problems chosen for exploration are the Macro Network Service, the Virtual MIB service and the Manager Delegation service. These address serious flaws in all the network management standards. In particular, solution for these will enable any computation (Macro) on any new MIB variable (Virtual MIB) and distribute these (Delegation) to any device in the network, automatically

The following table summarises the Active Networks Services produced in this research and the related Network Management problem identified in Chapter 2.

Active Networks Services	Network Management Problems
SNMP GetBulk Service	Inefficient protocols characteristics.
SNMP Threshold Service	Performance limitations of polling.
SNMP Macro Network Service	Lack of flexibility to update the system. Lack of a flexible system for defining generic event traps.
SNMP Virtual MIB Service	Lack of mechanisms to support adding new MIBs dynamically to a system. Dynamic MIB views.
SNMP Manager Delegation Service	Data Sharing among managers. Lack of scalability to manage a bigger system.

Table 3. Active Networks Services – Network Management problems

3.6 SUMMARY

This Chapter has presented Active Networks. Active Networks offer a more flexible network infrastructure. Briefly, Active Networks allow Active Nodes to perform computation on data; both the code and the data are carried in packets. A number of the

major Active Networks technologies were reviewed and it was argued that ANTS was the most appropriate for this research.

Section 3.5 selected the network management problems which would be solved using Active Networks.

Chapter 4

ENVIRONMENTAL MODEL FOR SNMP ACTIVE NETWORKS TOOLKIT

4.1 INTRODUCTION

The main objective of the SNMP Active Networks Toolkit is to provide a test-bed which will allow the standard SNMP system to be compared to the equivalent Active SNMP system; the services selected in Chapter 3 will be used as the basis for comparison. The structure of the test-bed is divided into two parts; these parts comprise the set of elements which provide the complete functionality for a standard SNMP network management system and the additional elements required for an Active Networks services network management system. The Toolkit links both parts so that the a comparison may be made between the standard SNMP approach to network management and the Active Networks SNMP approach to network management. The purpose of Chapter 4 is to explain the test-bed environment, in particular the technologies used to design and build the test-bed.

In the creation of both parts of the Toolkit a number of existing technologies have been employed; all use Java. As discussed in Chapter 3 ANTS was chosen as the Active Networks technology for the Active Networks SNMP part. The standard SNMP network management part was implemented using the Java Management Extension (JMX) [81] specification including the Java Dynamic Management toolkit (JDMK) [82], which is its reference standard implementation.

The Java Management Extension (JMX) is a Java standard specification that defines the way in which network management systems can be built. JMX defines interfaces and classes for all of the major network management standards, for example, TMN and SNMP. Clearly only the elements relating to SNMP were required for this research. As with every Java specification, a reference implementation is provided for JMX. This reference implementation is the Sun Microsystems product, Java Dynamic Management Kit (JDMK) and it was used in the creation of the Toolkit in preference to other licensed products. Both JMX and JDMK are discussed in more detail in Sections 4.2 & 4.3 respectively.

Currently, ANTS does not provide a means to automatically locate the Active Nodes where a known service is stored. ANTS only suggests using the standard procedures to obtain the network topology. After some research and experimentation it was discovered that this feature could be provided by a technology called Jini.[4] [83] Jini is also provided by Sun and is a Java based technology which is intended to simplify access to, and delivery of, an unrestricted range of network services. Essentially, it is a set of implementation guidelines, which will enable a federation of Java virtual machines to transparently respond to user network requests. In this research Jini is used to dynamically register all the Active Networks SNMP agent nodes.

Section 4.3 will consider Jini. In particular the Jini architecture will be discussed and the role it plays in the SNMP Active Networks prototype. Finally, section 4.4 will describe the design of architecture used to build the standard SNMP prototype and explain the roles JMX and Jini play in this architecture.

---

[4] Jini is a trademark of Sun Microsystems.

It is essential to understand that the standard SNMP network management system proposed for the toolkit is not a simulation, but a real SNMP system, with the MIBs in the network devices populated with real values. It is not however a full network management system in that not all the MIB variables are considered, hence the term prototype is used as well. The MIB values real values are found by periodically invoking network operating systems commands.

4.2 JMX

Java Management Extension (JMX) is a recent Java API specifically developed for creating network management systems. One very important property of any new network management development system, such as JMX, is that it should provide an easy means of integrating existing managed network devices into any new network management standards. Since most existing network devices are configured to use the SNMP standard, JMX has several features which are intended to make the integration of legacy SNMP devices relatively easy.

The origins of JMX can be traced to the original Java Management API (JMAPI) which was the first attempt by Sun to create a network management system using the Java language. For various reasons, including the fact that Java was, at that time, an immature language, JMAPI was less that a total success.

One of the main problems with JMAPI was that it did not take sufficient account of how network devices were currently being managed. [84] Any new management system which cannot properly accommodate legacy network devices is likely to have limited appeal.

Based partly on the early work with JMAPI, JMX was announced in July of 1999. JMX retains the flexibility and portability of the Java language but allows much easier integration of legacy network devices. JMX was developed specifically to target the network management standards CIM/WBEM, TMN and, most importantly, SNMP networks and the devices configured to work in them.

JMX has a three level architecture. Figure 6 shows the JMX architecture. Note that the word level does not imply a layered architecture, with one level above or below any other. Rather it signifies a different level of involvement in the JMX structure.

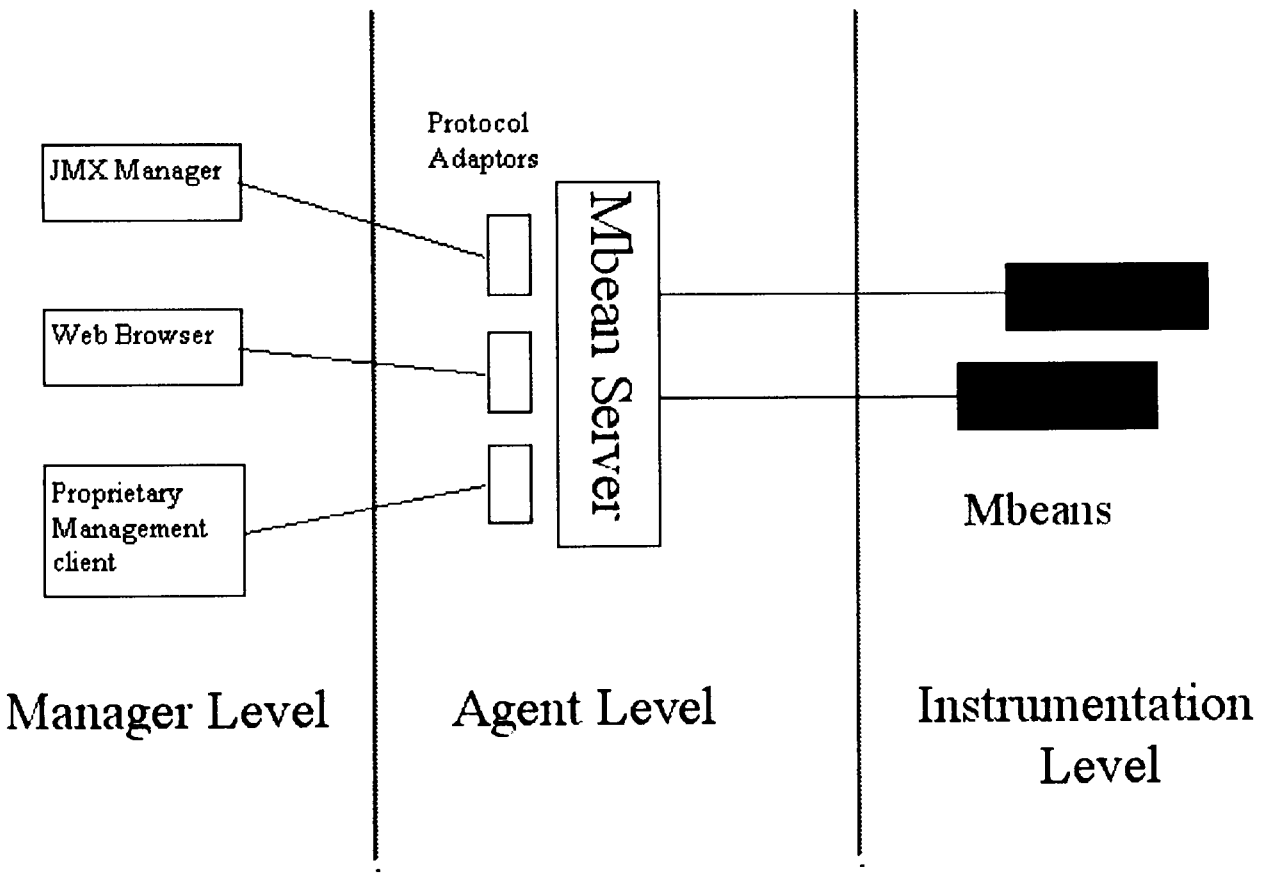


Figure 6. JMX Architecture.

4.2.1 THE JMX INSTRUMENTATION LEVEL

In JMX, every network device that can be managed is called an Mbean (management bean). Formally, an Mbean is defined as 'a Java beans design pattern compliant JMX resource that has passed the JMX Instrumentation Level Compatibility Test'. [85] The Instrumentation Level referred to in this formal definition is one element of the three levels JMX architecture. The JMX instrumentation level specification contains the rules for creating an Mbean. Mbeans can be applications, devices, services, or even users that are either constructed using the Java language or enclosed within a Java wrapper. Mbeans are Java objects. Consequently every Mbean, whatever its nature, can be accessed, manipulated and interrogated by other Java objects such as JMX agents or managers.

4.2.2 THE JMX AGENT LEVEL

The agent level specification in the JMX architecture states how agents should be implemented. The agent is the part of every managed device which controls one or more Mbeans in that device and makes it/them accessible to remotely management applications. It comprises an Mbean server, a set of services for handling Mbeans and at least one communications adaptor and connector.

The Mbean server is a registry for objects at the instrumentation level. In other words, Mbeans in a managed device must register with an Mbean server before they can be managed by a remote management application.

Protocol adaptors and connectors are the means by which this management is possible. Each protocol adaptor (and an agent may have more than one of these implemented)

provides an external view of Mbeans using a particular manager-agent protocol such as SNMP or HTML. In the case of SNMP for instance, the manager and agent can exchange SNMP commands and responses to control/monitor the Mbeans registered with the agent. The actual communication between manager and agent is by means of communication protocols such as HTTP, which is terminated in the JMX agent-level connector. The connector is a software component which responds to and initiates network messages.

4.2.3 THE JMX MANAGER LEVEL

This element of the JMX architecture specifies how JMX managers should be implemented. It defines the management interfaces and components that allow inter-action with JMX agents. Of particular interest in this study is the manager level specification for creating SNMP managers.

4.3 JDMK

The first implementation of JMX was included in the Java Dynamic Management Kit (JDMK). [82] As well as JMX, JDMK included a set of utilities, called tooling, which are not part of the formal JMX specification but could be used to build network management systems such as SNMP. For example, JDMK contains an SNMP Management Information Base (MIB) compiler and an API which simplifies the procedure for creating SNMP agents. It is the augmented version of the JMX contained within JDMK which is used in this SNMP implementation.

Basically, JDMK can be considered as a framework for the creation of Java-based management software and legacy SNMP-based systems using the JMX Standard.

4.3.1 JDMK ARCHITECTURE

The JDMK architecture can be divided in two main parts: Manager and Agent. The Agent part comprises the elements Core Management Framework, M-Beans and a number of different adaptors and services. The Manager part is composed of C-Beans and adaptors. Figure 7 summarizes the JDMK architecture.

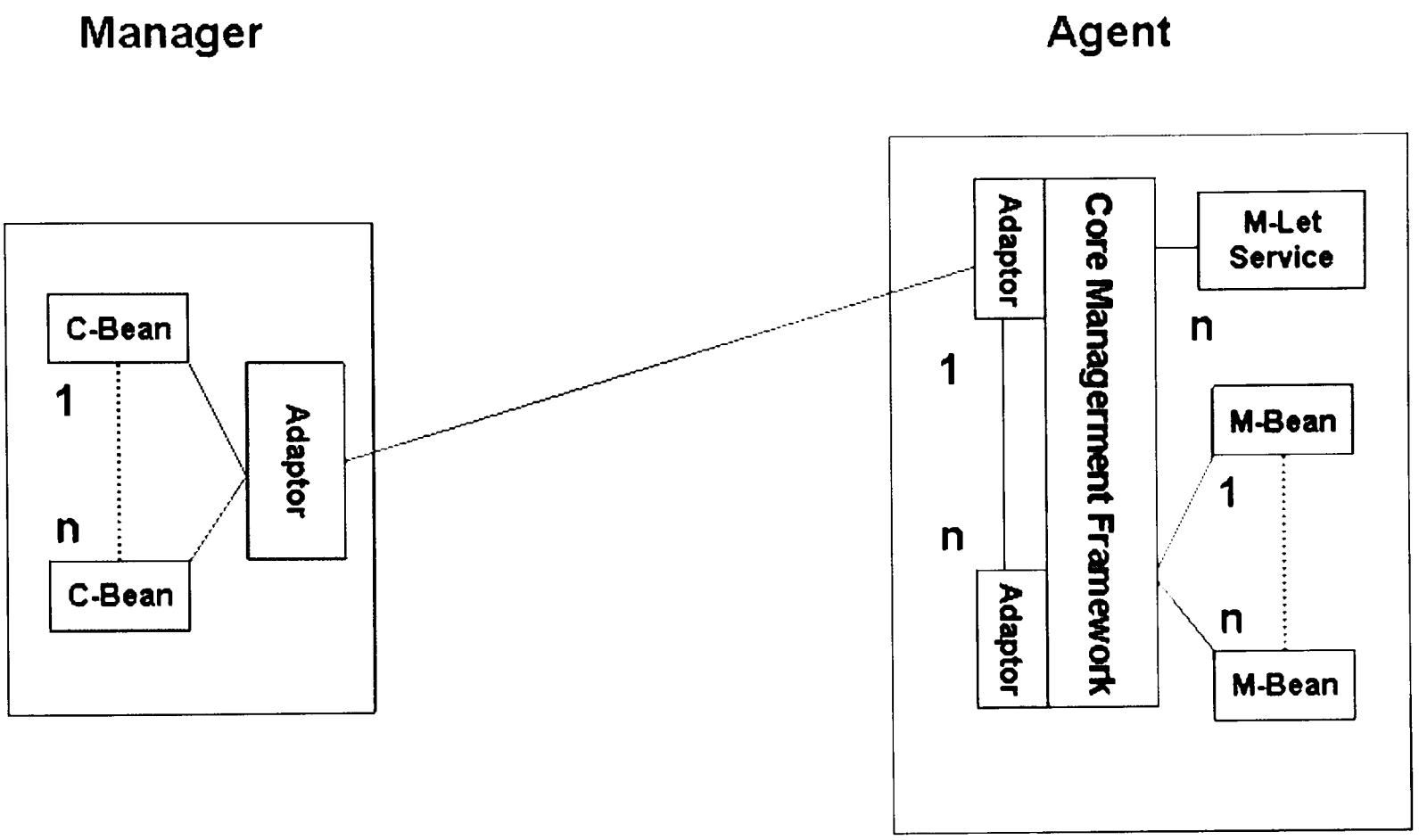


Figure 7. JDMK architecture.

A C-Bean (Client Bean) can be considered to be a proxy object for a remote M-Bean. Manager users can gain access to the M-Bean functions and Agent data by performing operations on C-Beans. Figure 7 illustrates how the C-Beans use adaptors to communicate with the remote M-Bean (Management Bean). Together with the adaptors they form the elements inside the Manager.

The Adaptor element implements a network protocol to allow communication between agent and manager. The Adaptor itself is a Bean which must be registered inside the Core Management Framework. Both Managers and Agents may implement multiple adapters and currently, JDMK provides adaptors for HTTP, HTTPS, IIOP, RMI, HTML and SNMP. This provides great flexibility as, for instance, a Manager can easily communicate with devices which are configured for different network management standards simply by implementing that standard. Clearly, this facilitates the building of a network management system which can effectively and simply integrate all existing network management standards and indeed allow new standards to also be accommodated.

In the Agent an M-Bean represents one or more managed objects. They are identified by a unique object name composed of the following parts:

class[.attribute=value[,attribute=value]*]. This unique identifier has the same purpose of the MIB object id: to provide a unique name for a network resource.

M-Beans have to be registered with the Core Management Framework to use the network resources needed for the Network Management communication. The Core Management Framework acts as repository of Mbeans and manages the network resources provided by the Adaptors to create an Agent.

Finally, JDMK provides a set of tools and services which complement its Network Management framework. One of these tools, Mibgen, is used to create the MIB needed in the SNMP system. Appendix B provides a more detailed analysis of JDMK and how it can be used to create a SNMP system.

4.3.2 USING JMX AND JDMK TO BUILD A SNMP NETWORK MANAGEMENT SYSTEM

JMX and JDMK provide a framework for building network management systems. They each comprise a set of Java interfaces, abstract classes which the developer must implement. At the time the SNMP network management system was developed no documentation existed to help the developer understand these complex frameworks. Thus, it was necessary to analyse the frameworks in detail, at the code level. This analysis can be found in Appendix B together with a class diagram.

4.4 ALTERNATIVES FOR BUILDING THE SNMP NETWORK MANAGEMENT TESTBED.

Early in the research, it became apparent that a SNMP network management test-bed would be needed. At this stage, JMX was not available; therefore, Jini was investigated as a possible platform for the test-bed. The intention was to use Jini as a distributed system which would pass SNMP commands between the network devices. Other distributed system technologies were considered, namely CORBA and Universal Plug and Play. CORBA was considered too complex, particularly for a network management test-bed. Universal Plug and Play was not mature at that time, but in any case, it was not Java based and therefore would not interoperate with the Java based ANTS. So Jini [105] was the preferred technology at this stage. However, when JMX and JDMK became available development with Jini stopped. JMX and JDMK were specific standards for building network management systems including SNMP systems. Therefore, these were clearly the best technologies to create the SNMP network management test-bed. However, later it

became apparent that Jini provided some interesting features which enhanced ANTS. In this research Jini was used to provide what can be described as a dynamic Domain Name System (DNS) service. The network devices (Active Nodes and non-active nodes) are registered with the Jini system. When the capsule must be sent to the next Active Node, Jini is given the name of the Active Node and it returns the corresponding IP address. The advantage of using Jini is that should an Active Node crash, it is automatically removed from the Jini server.

There are other ways in which Jini can enhance ANTS; these will be discussed in Chapter 7, Further Work.

4.5 JINI

4.5.1 INTRODUCTION

There are many definitions of Jini. Some of them classify it as a network operating system [83] and others as a technology that allows the creation of an impromptu community of consumer and network devices. [88] Formally, Jini is a framework that allows the development of distributed systems formed by 'devices' which support Sun's Java Development Kit (Java JDK 2).

From the user's point of view, Jini creates a federation of services connected by means of a network. Inside the federation any member has access to the exported features of all the other members. Important characteristics of Jini are the ease with which this federation can be created and the flexibility that exists to manage it. If a device is removed, the federation continues to operate as before; the whole process is transparent to the user. From the

programmer's point of view, Jini simplifies the process of programming distributed systems by providing transparent access to a network of computing resources rather than those of a single machine.

4.5.2 JINI ARCHITECTURE.

The Jini architecture [88] comprises:

- A group of components that provides the necessary infrastructure to create the federation of services mentioned above.
- A Programming Model that simplifies the process of creating distributed services.
- The Services themselves which are offered to and used by members of a federation. Typical examples of services are a computation, storage space, a communications channel, and a software filter or hardware device.

Figure 8 shows how the Jini architecture relates to the overall software architecture implemented in the various devices which make up a Jini-based system.

4.5.2.1 Jini Infrastructure.

The components which comprise the Jini infrastructure are,

- The Discovery and Join protocols which are supported by all Jini devices[5] and allow them to join a federation and offer their services to other members.
- One or more Lookup Servers which hold details of all current members of a federation.

[5] A Jini device is one that followed the standard Jini specification

- A Security System which is distributed between the devices and lookup servers in the Jini network.

A brief description of each of these components should explain more clearly the Jini infrastructure.

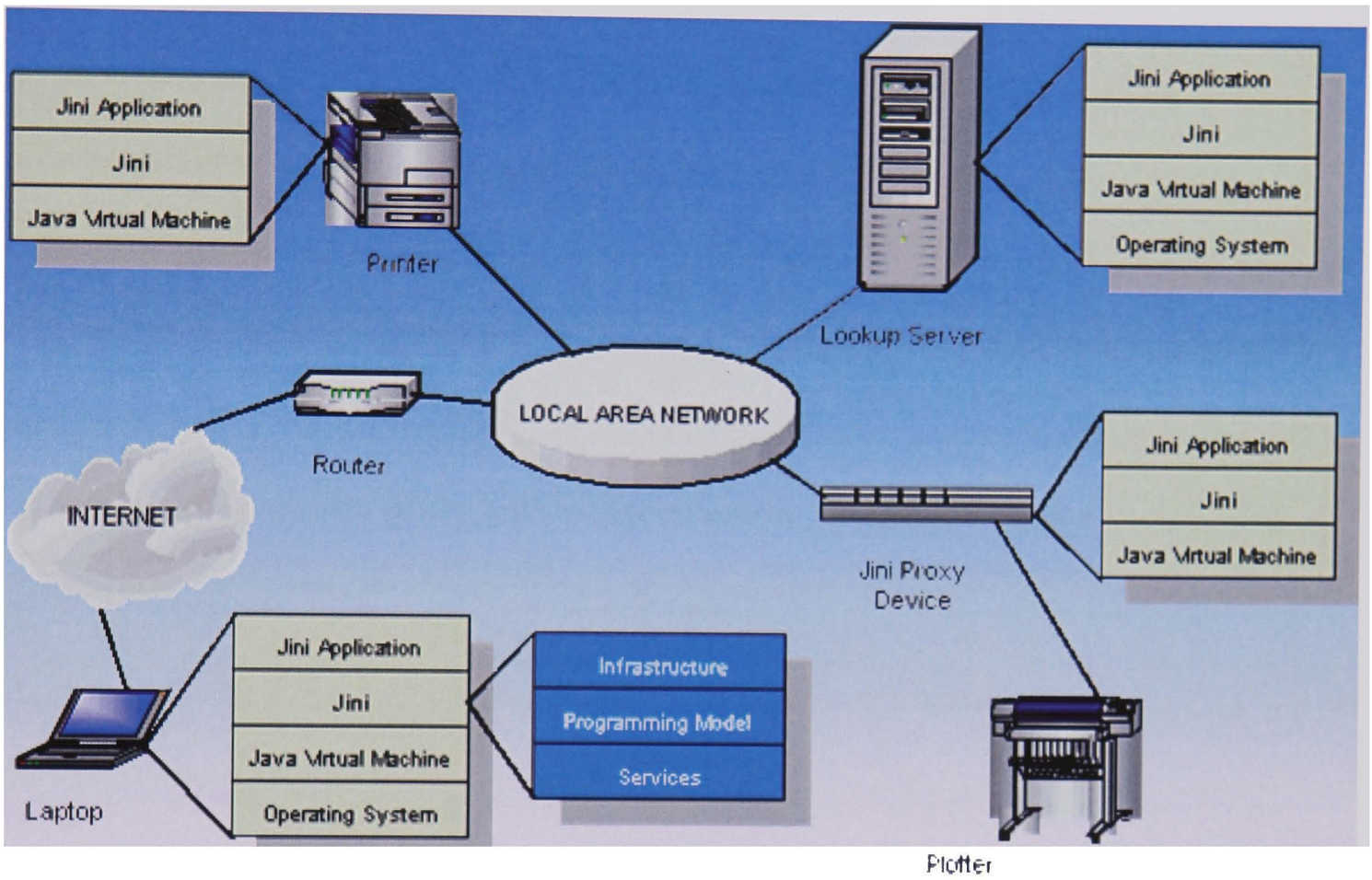


Figure 8. A typical Jini system.

4.5.2.1.1 The Discovery and Join protocols.

Essentially, the Discovery protocol is used by a newly initialised Jini device to discover the location of all the lookup servers in the environment and the Join protocol is used to register with them. After that, the newly enrolled member of the federation uses the Lookup protocol to locate other services which are needed for some networking task or distributed application.

In the discovery phase the new service multicasts a request for all lookup servers in the network to identify themselves. Having discovered one or more lookup servers, the new service (the service provider) then registers with the lookup server(s) - i.e., joins the federation - by providing a proxy object which is deposited with the lookup server. This proxy object describes the functionality of the new service and how it should be accessed. A client looking for a particular service uses the lookup protocol to access the lookup server and acquire a copy of the appropriate proxy object. The proxy object initiates communication between the service provider and the client and, depending on the nature of the service, the service may be provided locally or remotely. Communication is normally carried out using Remote Method Invocation (RMI) but this is not mandatory. Although more complex scenarios are accommodated in the Jini infrastructure, the essence of the protocols is as described.

4.5.2.1.2 The Lookup Server.

The Lookup Server is a service based on the 'Linda co-ordination model using Tuple Spaces' project at the Yale University [89] and is itself, the main service in a Jini System. As such, a Lookup Server can register with other Lookup Servers allowing the creation of several kinds of topological structures, for instance, hierarchical and mesh topologies. Sun [88] claims that it is even possible to enter a Lookup Server entry in a directory or name service and thereby implement an unrestricted WAN Jini system.

4.5.2.1.3 The Distributed Security System.

The Distributed Security System in Jini is based on the security enhancements which are included in the most recent version of the Java programming language. [90] In effect, these

can limit the groups of services that a client or a service can discover. However, there are several Jini security issues which, currently, have not been addressed. One of the most crucial is the use of Jini through firewalls and another is the use of Jini with the Secure Sockets Layer (SSL) protocol. Also, authentication is not carried out on discovery requests or responses. This means that it is impossible to know whether the entity being contacted is a 'legitimate' service or not. Additional security measures, beyond those already specified for Java, were not considered when Jini was originally conceived, but new technologies, such as the Java Authentication and Authorisation Service (JAAS), and, perhaps more important, the new RMI Security Extension, will probably be in the next version of Jini.

4.5.2.2 The programming model.

The Jini programming model comprises a number of standardised interfaces which characterise the main features of a fully-configured Jini system. These features are adequately described in the literature [91, 92, 93]. These interfaces are based on the Java programming language model but extended to take into account the distributed (network based) environment; the complications of delay and the uncertainties of the underlying network are taken into account in Jini. The publication of these interfaces is intended to facilitate the construction of new reliable services which conform to the Jini system specification.

4.5.2.3 Services.

Any member of a Jini system is a Service; the Infrastructure and the Programming Model are there only to provide a base on which to build services. A Service is a Java program inserted in a Jini compliant device, which will permit the control and use of the device in

the Jini system. Services therefore appear to be Java objects. They have interfaces which specify their functionality and these can be used by other programs or provide the means by which the service interacts directly with the user.

4.5.3 Building a Jini System

Similar to JMX and JDMK, Jini is a framework which defines interface and abstract classes which must extended by the developer. However there was documentation provided with Jini which was sufficient to develop a Jini system. Therefore, more details are not provided in this thesis.

4.6 SUMMARY

The purpose of this Chapter has been to present the technologies used to create the SNMP Active Networks Toolkit. To make the Toolkit compliant with the SNMP protocol, the Java standard JMX and its default implementation JDMK has been used. The Chapter has introduced JMX and JDMK, with special emphasis in their SNMP features. Both technologies provide the framework to create a complete SNMP system. The Toolkit also makes use of Jini as the means with which to maintain the changing Active Networks topology. The dynamic nature of the Active Networks topology matches perfectly with the Jini 'federation' concept. The next Chapter will analyse the SNMP Active Networks Toolkit architecture, how it operates and the procedure used to obtain real network management data.

Chapter 5

THE ARCHITECTURE FOR STANDARD SNMP AND ACTIVE SNMP NETWORK MANAGEMENT TOOLKIT

5.1 INTRODUCTION

The research posed in Chapter 1 clearly had a major effect on the design of this test-bed. In particular, two network management systems were needed, the standard SNMP system, which operates as a traditional SNMP network management system, and an Active SNMP system which uses the ANTS technology. [106] An incremental approach was also adopted in the development of the test-bed. First, the standard SNMP system was developed, using the JMX/JDMK technologies; the design of this is presented in Chapter 5.2. To provide the Active SNMP system the ANTS system and Jini had to be integrated with the standard SNMP system; this is discussed in Section 5.3. The Active SNMP prototype is essentially the standard SNMP version augmented with 2 extra menu options Active Node and Active Extension. An Active Network service may require the use of one or more extensions. The standard ANTS system only provides the functionality to load these extensions statically. This research has proposed a way in which extensions can be loaded dynamically, on demand, by any service and this is also included in Section 5.3.

Note that both systems are real and complete SNMP systems, which stores and retrieves real data from network devices within a real network. This is achieved by using operating system commands. They are however prototype systems because only a subset of the MIB

variables are considered and no the statistical analysis of the MIB information, typically provided by network management systems, is given.

5.2 THE STANDARD SNMP SYSTEM

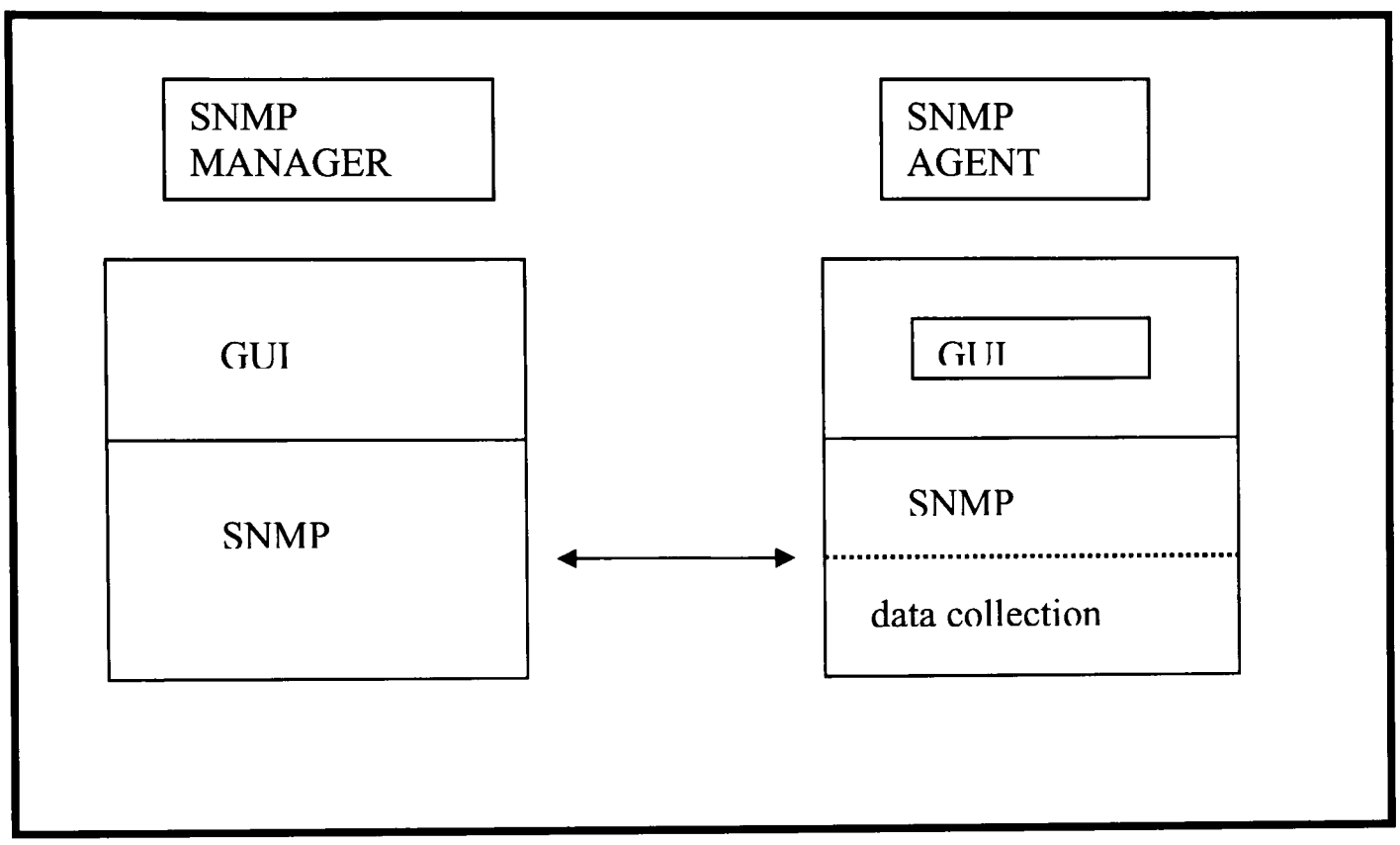


Figure 9. SNMP system structure.

Figure 9 shows the architecture of the standard SNMP system. As with any standard SNMP system, its structure is divided in two parts, manager and agent. The manager component is responsible for retrieving information from the agents and issuing commands to them using the standard SNMP commands. The prototype provides a graphics user interface (GUI) which emulates the network console typically used by the human network manager to interact with the network.

The agent provides the functionality to control a network device. This control is achieved by setting and retrieving the information stored in the agent SNMP MIB, within the network device. In the prototype, the MIB used is the standard SNMP MIB: MIB-II (RFC 1213). [95] Figure 10 shows the relation between the output of the two operating system commands, 'netstat-s' and 'netstat-a', with specific groups in the standard MIB: MIB-II. These are the commands use to populate the MIB-II with real data.

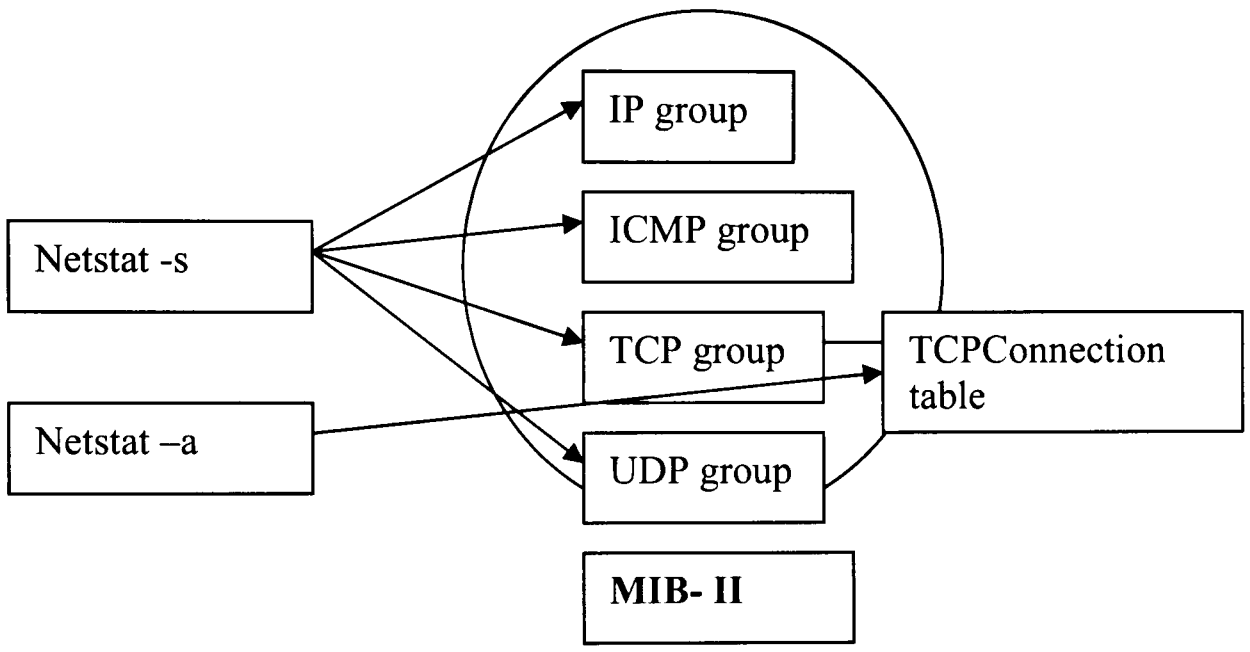


Figure 10. Relation between SO commands and MIB-II.

The next section will present the standard SNMP system architecture in more detail.

5.2.1 SNMP MANAGER

The SNMP manager is composed of two layers: A GUI which the human network manager uses to interact with the network and a SNMP layer that transforms the commands given into standard SNMP commands.

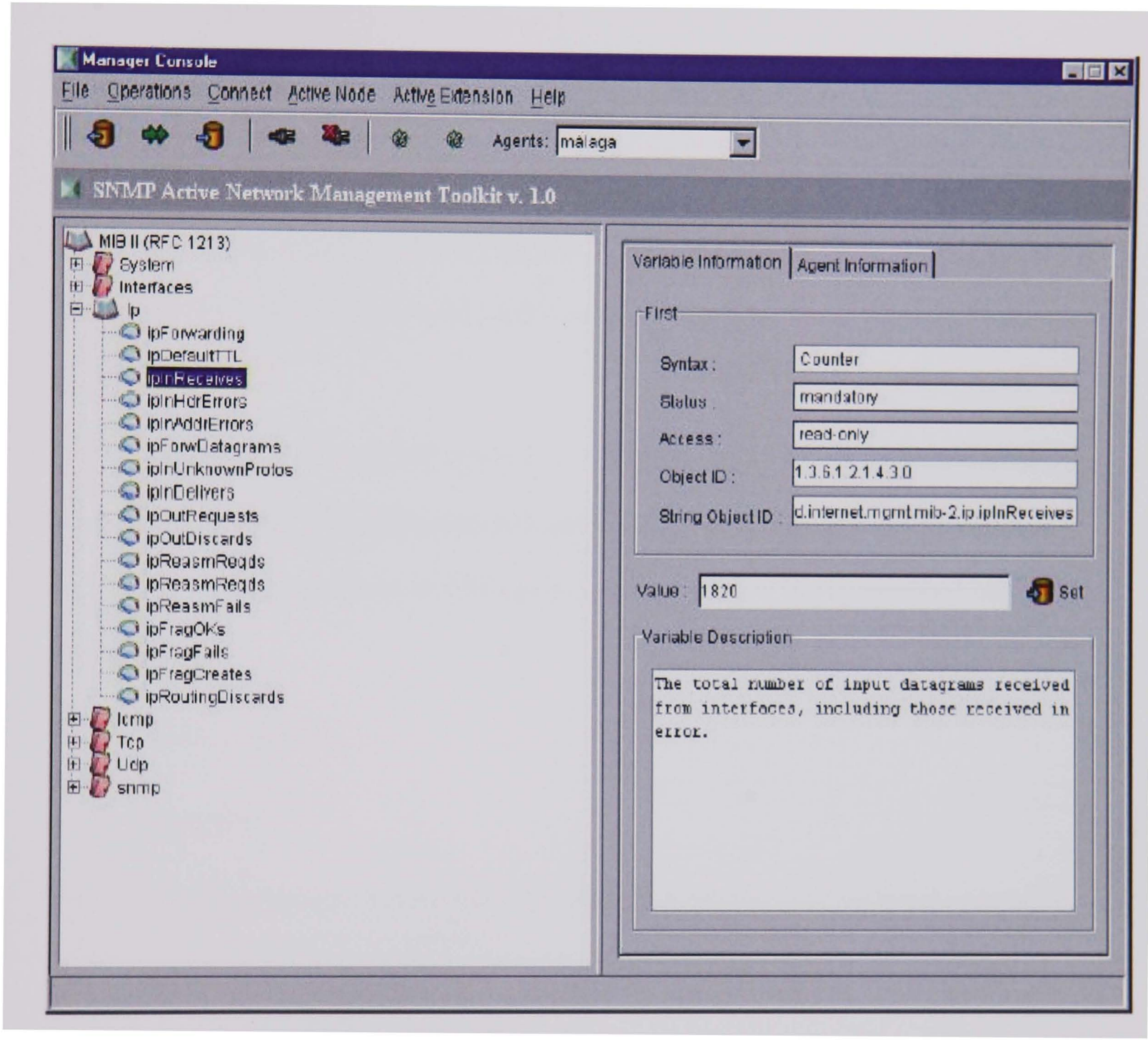


Figure 11. The Network management console.

5.2.1.1 The GUI layer.

The network management console GUI design is shown in Figure 11 for the standard SNMP system. This layer is divided in three areas:

- Menu area
- Toolbar area
- MIB area.

The menu area has a number of menu options which group a common set of commands. For the standard SNMP system menu options File, Operations, Connect and Help are provided.

The option File allows the SNMP manager to be exited, the option Operations includes the main SNMP commands GetRequest, SetRequest, GetNextRequest and GetBulkRequest and the option Connect allows SNMP agents to add or deleted. The information needed to add an agent is:

- Agent IP address.
- Agent port.

The toolbar area contains icons as shortcuts for the options found in the Menu area and a combo box element is provided to switch between the different SNMP agents which are currently part of the network. For instance, Figure 11 shows that Malaga is the name of the agent which the next SNMP command will be sent to.

The MIB area has two main functions:

- To allow the user to select a MIB variable; this is needed to create an SNMP command.
- To show the result of an executed SNMP command.

The area is divided in two sections: a MIB tree section (left hand side) and an information section (right hand side). The MIB tree shows a tree structure matching the MIB-II structure. Figure 12 shows the MIB-II tree structure relating to network nodes which are hosts, for example computers. When the user clicks one of the MIB Variables in the tree structure the GUI layer sends a GetRequest command to the SNMP layer with the information associated with the selected MIB Variable. The SNMP layer, using the JMX/JDMK API, sends a SNMP GetRequest command to the agent currently selected in the GUI (which is Malaga shown in Figure 11). The response from the agent is received by the manager, specifically the SNMP layer. The SNMP layer analyzes the results and sends the appropriate information to the GUI layer. Once the results are received by the GUI layer, they are presented in the MIB area information section for viewing.

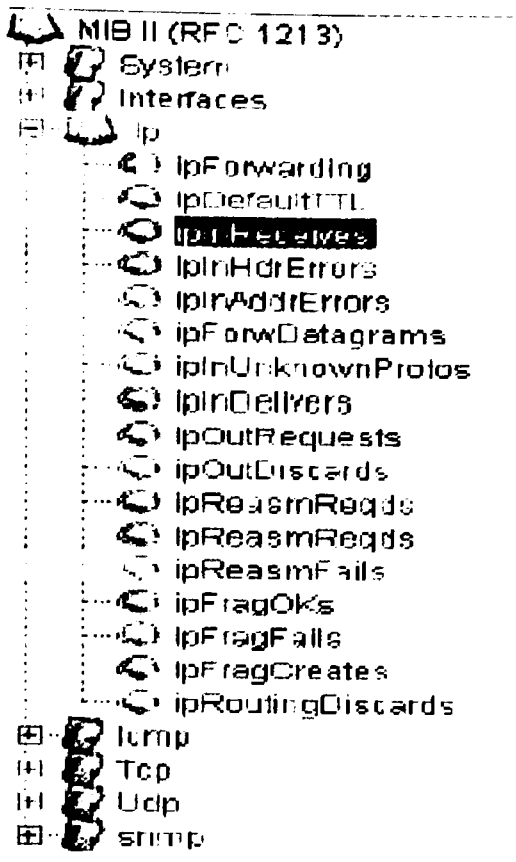


Figure 12. MIB-II nodes tree.

The MIB area information section (right hand side) is used to view information relating to:

- MIB Variable currently selected
- SNMP agent currently selected.

These are not viewed concurrently, instead the user can choose which to view using the two tabs Variable information and Agent information. Figure 11 show the Variable information tab used to view the information relating to the MIB Variable IpInReceives.

With the exception of the Value property, the information shown is defined precisely by the MIB-II standard and held as read-only properties in the MIB. The properties defined for each MIB Variable are;

- Syntax. This is the name of the SNMP type.

- Status. Indicates information about whether the information is mandatory in the agent or not.
- Access. Indicates if the content can be changed or not by the manager.
- Object ID. Shows the unique id.
- String Object Id. Shows the unique string id.
- MIB Variable description. Shows a description of the functionality.

The Value property is the only one which can be updated in the MIB by the user, using the SetRequest command, and subsequently retrieved from the MIB using the GetRequest command. Clearly the access property dictates whether a MIB Variable can be updated. The button Set must be used to send the 'SNMP set' command to the agent. As any GUI action, the SNMP layer receives the action and transforms it to the correct SNMP command and executes it.

In the case of an Agent, the information stored will be the Agent address (the IP address) and the Agent port number. The information shown about an Agent relates to Agent currently selected by the user; the values shown in Figure 11 relate to the Malaga Agent.

5.2.1.2 The SNMP layer

The SNMP layer is composed of a number of classes whose function is to receive the commands from the GUI layer and to transform them into one or more standard SNMP commands, using the JMX/JDMK API. The main class for this layer is called InnerManager. The class, whose signature is shown in Figure 13, has a method for every type of GUI layer command.

```
public void getRequest(MibNode node,String action)

public String[] getNextRequest(String variableRequest)

public void getBulk(int maxRep)
```

Figure 13 Public InnerManager Methods.

Each of the methods shown in Figure 13 is discussed below.

- public void getRequest(MibNode node,String action) throws InnerManagerException.. This method is used by the GUI layer commands 'get' and 'set'. The action parameter is used to decide precisely which SNMP command the GUI command relates to. The MIB Variable selected (highlighted) by the user is automatically the target variable for this command and is the node parameter in the method signature.

- public String[ref] getNextRequest(String variableRequest) throws InnerManagerException.. This method is used by the GUI layer command 'GetNext' to produce the corresponding SNMP command. The GUI layer uses the MIB Variable selected from the MIB Tree, and retrieves the next MIB Variable in the MIB Tree.

- public void getBulk(int maxRep) throws Exception .This method is provided by the layer as a specific aspect of the research and when invoked it accesses one table in the MIB – hence the absence of a parameter identifying the table to be accessed.

The MIB table used is always the MIB-II table 'TcpConnTable'. The main object of this method is to investigate the performance of accessing table of various sizes.

All these methods use the JMX/JDMK API to create the corresponding SNMP commands to be sent to the SNMP agent.

5.2.2 SNMP AGENT

The SNMP Agent system provides the required functionality of a standard SNMP agent. As noted earlier, SNMP Agents are deployed in real network devices (host computers), and populated with real data, updated periodically.

The SNMP agent is composed of two layers: A GUI layer which is used to start and stop the agent and a SNMP layer that has two main functions:

- Receive and respond to SNMP commands sent by a manager. The commands are received on a standard port where the agent is always listening.
- Periodically populate the MIB with data. Clearly MIB used is identical to the one used by the SNMP manager.

5.2.2.1 The GUI Design

The GUI is a small window, with two buttons. The top button allows the SNMP Agent to be initiated and the lower button allows the SNMP Agent to be terminated. The process of starting an agent is divided internally into a number of steps:

- Initialising static Agent MIB Variables. Some MIB variables are static by nature, e.g. SysLocation, which is defined in the System group in the MIB. The values for

these MIB Variables are initialised from information held in a file, when the system is started. This must be done first.

- Starting SNMP agent layer. This starts the procedure which completes the initialisation of the dynamic MIB variables, and finishes with the agent listening at the standard SNMP port.

- Starting data collection system. The system integrated in the SNMP agent layer creates a thread that periodically executes the 'netstat –s', 'netstat –a' operating system commands. The results from these commands are used to populate the MIB.

5.2.2.2 The SNMP agent layer.

The SNMP agent layer is composed of two integrated but separate subsystems:

- The SNMP agent subsystem.
- The data collection subsystem

The SNMP agent subsystem is analogous to the SNMP manager. The SNMP agent system waits for a SNMP command from a SNMP manager. Once a command has been received and validated (the agent checks that the manager has permission to execute the command by checking the Access Control List which is held in a Java property file), the Agent executes the command and an SNMP response is returned to the manager.

The command will entail accessing the MIB and returning various information as part of the response.

The data collection subsystem works independently from the SNMP Agent. This subsystem is initiated as a thread and its sole function is to execute operating system commands. The output of these commands is processed by a parser which extracts the data matching the corresponding MIB-II variables.

```
C:\Documents and Settings\Antonio>netstat -s

IPv4 Statistics

  Packets Received                   = 110
  Received Header Errors             = 0
  Received Address Errors            = 6
  Datagrams Forwarded                = 0
  Unknown Protocols Received         = 0
  Received Packets Discarded         = 0
  Received Packets Delivered         = 110
  Output Requests                    = 13595
  Routing Discards                   = 0
  Discarded Output Packets           = 0
  Output Packet No Route             = 0
  Reassembly Required                = 0
  Reassembly Successful              = 0
  Reassembly Failures                = 0
  Datagrams Successfully Fragmented  = 0
  Datagrams Failing Fragmentation    = 0
  Fragments Created                  = 0

ICMPv4 Statistics

                        Received    Sent
  Messages              1           1
  Errors                0           0
  Destination Unreachable 1         1
  Time Exceeded         0           0
  Parameter Problems    0           0
  Source Quenches       0           0
  Redirects             0           0
  Echos                 0           0
  Echo Replies          0           0
  Timestamps            0           0
  Timestamp Replies     0           0
  Address Masks         0           0
  Address Mask Replies  0           0

TCP Statistics for IPv4

  Active Opens                       = 13481
  Passive Opens                      = 1
  Failed Connection Attempts         = 13460
  Reset Connections                  = 2
  Current Connections                = 0
  Segments Received                  = 20
  Segments Sent                      = 13497
  Segments Retransmitted             = 6

UDP Statistics for IPv4

  Datagrams Received    = 65
  No Ports              = 22
  Receive Errors        = 0
  Datagrams Sent        = 89
```

Figure 14.Netstat –s output.

The operating system commands which are executed are:

- 'netstat –s'. This command has as output, network statistics sorted by Internet protocol. By default, statistics are shown for the network protocols: IP, ICMP, TCP, and UDP. Figure 14 shows a normal output for this command. It is clear from the Figure that almost all the statistics obtained from the output of this operating system command are the values of MIB-II variables.

```
C:\Documents and Settings\Antonio>netstat -a

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    malaga:epmap           0.0.0.0:0              LISTENING
  TCP    malaga:microsoft-ds    0.0.0.0:0              LISTENING
  TCP    malaga:1025            0.0.0.0:0              LISTENING
  TCP    malaga:1030            0.0.0.0:0              LISTENING
  TCP    malaga:4595            0.0.0.0:0              LISTENING
  TCP    malaga:4596            0.0.0.0:0              LISTENING
  TCP    malaga:4597            0.0.0.0:0              LISTENING
  TCP    malaga:4598            0.0.0.0:0              LISTENING
  TCP    malaga:4599            0.0.0.0:0              LISTENING
  TCP    malaga:4600            0.0.0.0:0              LISTENING
  TCP    malaga:4601            0.0.0.0:0              LISTENING
  TCP    malaga:4602            0.0.0.0:0              LISTENING
  TCP    malaga:4603            0.0.0.0:0              LISTENING
  TCP    malaga:4604            0.0.0.0:0              LISTENING
  TCP    malaga:4605            0.0.0.0:0              LISTENING
  TCP    malaga:4606            0.0.0.0:0              LISTENING
  TCP    malaga:4607            0.0.0.0:0              LISTENING
  TCP    malaga:4608            0.0.0.0:0              LISTENING
  TCP    malaga:4609            0.0.0.0:0              LISTENING
  TCP    malaga:4610            0.0.0.0:0              LISTENING
  TCP    malaga:4611            0.0.0.0:0              LISTENING
  TCP    malaga:4612            0.0.0.0:0              LISTENING
  TCP    malaga:4613            0.0.0.0:0              LISTENING
  TCP    malaga:4614            0.0.0.0:0              LISTENING
  TCP    malaga:4991            0.0.0.0:0              LISTENING
  TCP    malaga:5000            0.0.0.0:0              LISTENING
  TCP    malaga:8888            0.0.0.0:0              LISTENING
  TCP    malaga:netbios-ssn     0.0.0.0:0              LISTENING
  TCP    malaga:4595            169.254.124.101:epmap  SYN_SENT
  TCP    malaga:4596            169.254.124.102:epmap  SYN_SENT
  TCP    malaga:4597            169.254.124.103:epmap  SYN_SENT
  TCP    malaga:4598            169.254.124.104:epmap  SYN_SENT
  TCP    malaga:4599            169.254.124.105:epmap  SYN_SENT
  TCP    malaga:4600            169.254.124.106:epmap  SYN_SENT
  TCP    malaga:4601            169.254.124.107:epmap  SYN_SENT
  TCP    malaga:4602            169.254.124.108:epmap  SYN_SENT
  TCP    malaga:4603            169.254.124.109:epmap  SYN_SENT
  TCP    malaga:4604            169.254.124.110:epmap  SYN_SENT
  TCP    malaga:4605            169.254.124.111:epmap  SYN_SENT
  TCP    malaga:4606            169.254.124.112:epmap  SYN_SENT
  TCP    malaga:4607            169.254.124.113:epmap  SYN_SENT
  TCP    malaga:4608            169.254.124.114:epmap  SYN_SENT
  TCP    malaga:4609            169.254.124.115:epmap  SYN_SENT
  TCP    malaga:4610            169.254.124.116:epmap  SYN_SENT
  TCP    malaga:4611            169.254.124.117:epmap  SYN_SENT
  TCP    malaga:4612            169.254.124.118:epmap  SYN_SENT
  TCP    malaga:4613            169.254.124.119:epmap  SYN_SENT
  TCP    malaga:4614            169.254.124.120:epmap  SYN_SENT
  UDP    malaga:epmap           *:*
  UDP    malaga:microsoft-ds    *:*
  UDP    malaga:isakmp          *:*
  UDP    malaga:1026            *:*
  UDP    malaga:ntp             *:*
  UDP    malaga:1900            *:*
  UDP    malaga:ntp             *:*
  UDP    malaga:netbios-ns      *:*
  UDP    malaga:netbios-dgm     *:*
  UDP    malaga:1900            *:*
```

Figure 15. Netstat –a output.

- 'netstat –a'. This command has as output all the connections together with the ports which the connections are listening on. The output of this command will provide the data to populate the TCPConnection table. The Figure 15 shows a typical output of this command.

Note that these two operating system commands get their output directly from the network card plugged into the Agent network devices; the Agent devices in this test-bed are hosts, in this case PCs.

5.3 THE ACTIVE SNMP SYSTEM

This section describes the additional functionality needed to augment the standard SNMP system and thus create the Active SNMP version. The changes can be classified into two categories:

- GUI.
- Functional.

The standard SNMP prototype GUI must be enhanced to offer menu options to initialize and start the Active network management services. These new graphics user options correspond to the additional functional requirements needed for the Active SNMP system. The additional GUI alterations affect only the manager, while the functional changes affect both manager and agent. The next sub-sections give specific details about these enhancements.

5.3.1 GUI ENHANCEMENTS

The additions allow a user to configure and invoke an Active network management service.

Figure 16 shows the two additional menu options.

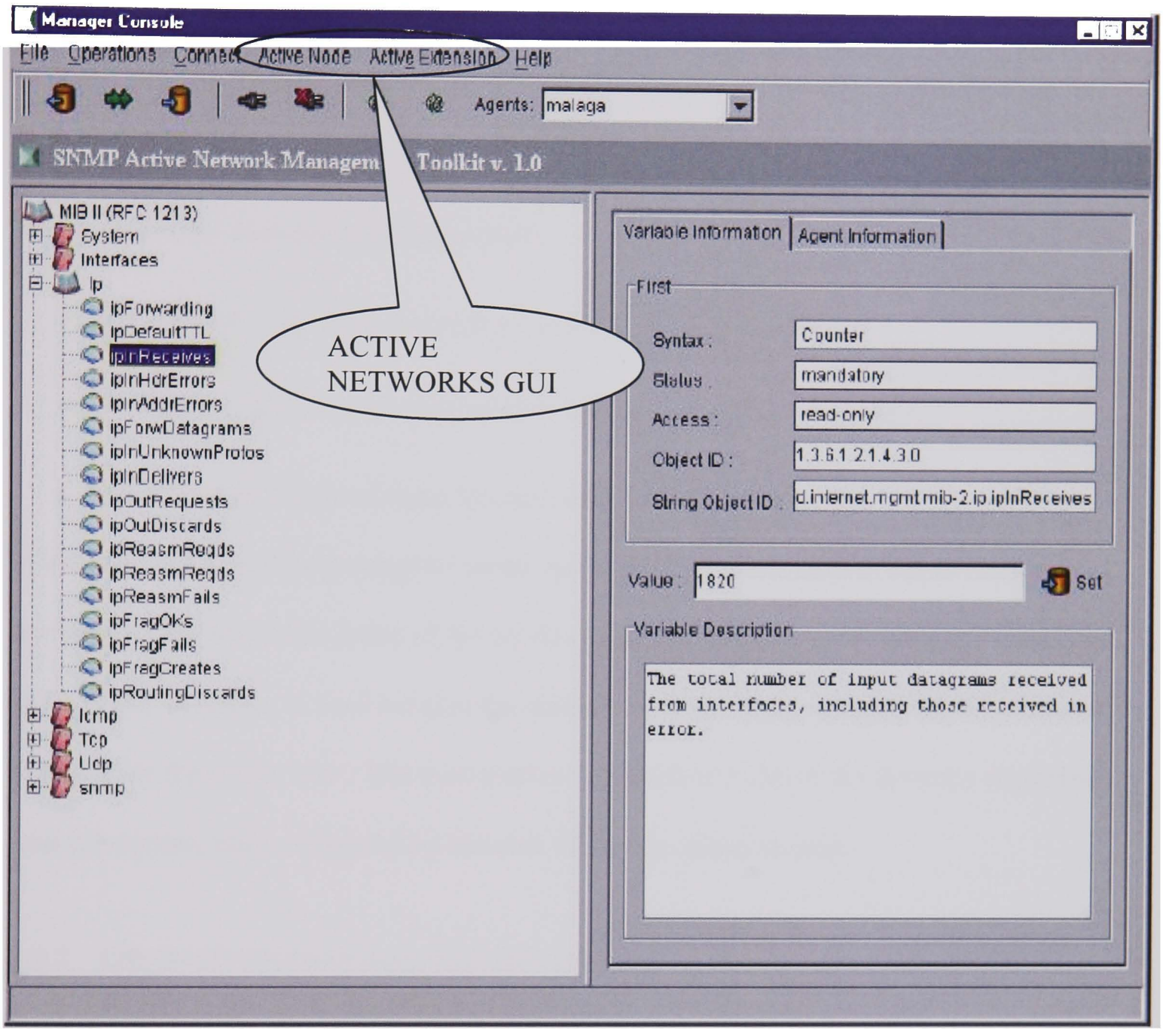


Figure 16. The Active Network management console.

The first Active SNMP option is called 'Active Node' and it comprises a number of Active Network node configuration options, together with the mechanism to start the Active

network management service. The second Active SNMP option, called 'Active Extension', provides the means to load ANTS service extensions. Both options are interrelated. An ANTS service is typically composed of a number of extensions that must be installed in the network before the service can be executed. The extensions are installed by using the Active Extension menu option, and then the Active Node menu option can be used to execute the service.

The Active Node configuration options are:

- Source Port. Sets the port number for the connection source.
- Destination Port. Set the port number for the connection destination.
- Node Target. The host name for next Active Node in the path.

In the case of the 'Active Extension' menu option, the only information which must be provided by the user is the name of the service to be executed; this indirectly provides the name of the extension to load because the extension name is composed of the service name followed by the suffix 'Ext'. The next section will explain in detail the dynamic extensions load sub-system that is triggered by the user when this option is used.

5.3.2 DYNAMIC EXTENSIONS LOAD SYSTEM

Before a service can execute in an Active Node, all extensions which it requires must be present in the node. ANTS provides the functionality for services to make use of extensions, but it does not provide the means to dynamically load the extensions into Active Nodes. As part of this research a new dynamic extensions load system was designed and implemented. The main goal of this dynamic load system was to distribute the

extensions needed by a service to all the Active Nodes visited by the service. Figure 17 shows how this distribution takes place.

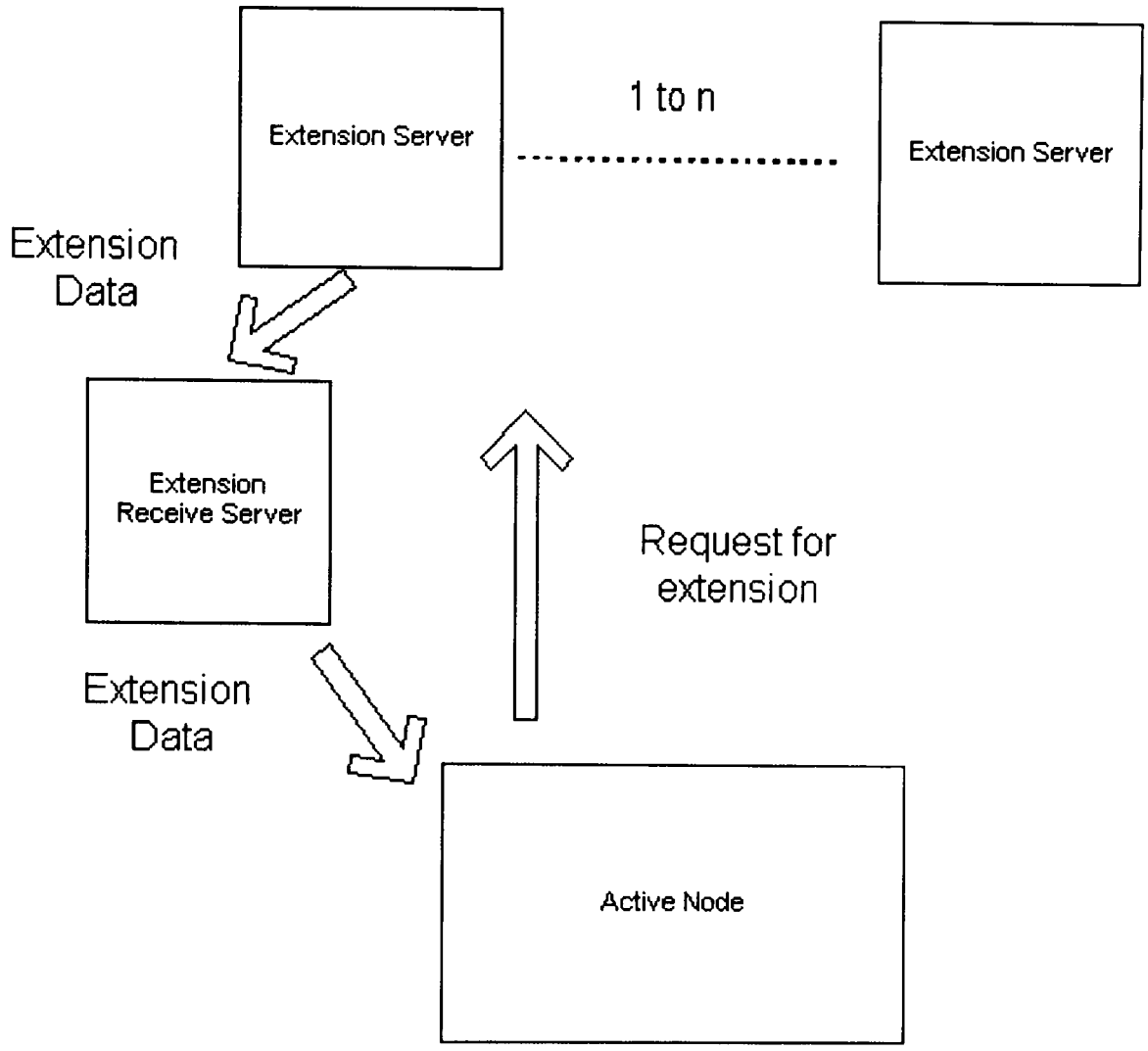


Figure 17. Dynamic extensions load system.

The system is a typical client-server system where the data belonging to the service extension is sent, serialized[6] from one Active Node to another. Once the data is received in the target node, the service extension class is reconstructed. The name of the class is composed of the name of the service followed with the suffix 'Ext'. The sequence of operations required to achieve the dynamic loading of an extension are as follows, and corresponds to the actions in Figure 17.

[6] A java language concept used actively to move objects in the network.

1 A number of Extension servers are started. The extension server's main task is to wait, listening for requests asking for active network extensions. A request will provide the name of the extension to be sent. The extension servers must have access to the extensions needed by the services. In this research the extension servers have been loaded with the service extensions manually. Clearly, an automatic procedure to sent new extensions to all the extension servers is needed.

2 An Extension worker server is started as a separate thread by every Active Node, as part of the Active Node initialisation procedure. The extension server has the same IP address as the Active Node and uses the same port number in all the Active Nodes. This server is waiting all time for data from the extension server.

3 A request is made to load an extension. This request will send the name of the extension required to an extension server. The information about which extension server to use has been set statically, as part of the initialisation procedure.

4 The extension server receives the request and sends, as a response, the extension class requested, serialized, to the Node Extension worker server.

5 The Active Node reconstructs the extension class with the information received by the Extension worker server, from the Extension Server.

6 The new extension class is now registered in the Active Node.

The way in which the Dynamic Extension Load System works is similar to how a plug-in is installed in a browser, for example. As with plug-ins, the extension is downloaded and registered once in the Active Node. The extensions have the purpose of extending the

Active Node functionality that can be used by the services. When an extension is created, the concept of it being generic and thus useful for many services, is desirable. The service creators must analyze the extensions already available in the Active Nodes before the creation of the service. Functionality which is not considered generic must be created as Capsules. A standard library of extensions could solve the problems of reutilization of code among the services. This library could be also optimized to be as fast and small as possible with the purpose of being used by a larger number of Active Nodes.

5.4 SUMMARY

The Chapter has presented the architecture of the SNMP Active Networks Toolkit. To ensure that the toolkit is compliant with the SNMP protocol, the standard JMX and its default implementation JDMK has been used. The Toolkit also makes use of Jini as the mean to maintain the changing ANTS topology information. Jini is a framework to develop distributed systems where every member has access to all the services provided by the others.

Chapter 6

ANALYSIS AND EVALUATION OF NETWORK MANAGEMENT SERVICES

6.1 INTRODUCTION

Chapter 5 presented the standard SNMP test-bed for both the standard SNMP systems and the Active SNMP system. This Chapter uses the test-bed to design and build the selected services which will be used to compare the standard SNMP network management system with the equivalent Active SNMP network management service.

Section 6.2 first explains the design pattern used to deploy Active Network services.

Section 6.3 is by far the most important section in this Chapter. It presents a detailed comparison of the candidate services (selected in Chapter 3) in both the standard SNMP and the Active SNMP systems. A quantitative analysis is available for the specific SNMP services (GetBulk and Threshold) but this is not appropriate for the remaining 3 services as these can only be made available in the Active SNMP system. It is argued that these 3 services offer a solution to a number of fundamental operational problems shared by all network management systems. Furthermore, these 3 services provide a new network management paradigm for future network management systems.

6.2 ACTIVE SNMP NETWORK MANAGEMENT SERVICES

The creation of Active Network Services requires two main tasks to be carried out:

- The analysis, design and creation of the classes that compose the service.
- The deployment of these service classes into the Active Network.

The creation of Active Network services in ANTS follows a design pattern that must be used by all the Active services. This design pattern was proposed by Wetherall [11] and is comprised of a number of operations that every service must execute. The next section will explain in detail how this design pattern is used to create an Active Network service.

6.2.1 ACTIVE NETWORK SERVICES DESIGN PATTERN

ANTS provides an API with a number of classes to the service developer to help create an Active Network service. There are a number of classes which the developer must extend, as follows:

- Application.
- Capsule.
- Protocol.
- Extension.

In Chapter 3, an overview of these classes was presented, and a detailed discussion is given in Appendix A. An ANTS service is an Application running in the ANTS Active Network,

composed of a number of Capsules, which are grouped in a number of Protocols, and which use a number of Extensions. Figure 18 shows the methods that must be extended for these four classes.

Class	Method
Capsule	execute
	encode
	decode
Application	start
	receive
Protocol	addCapsule
Extension	action

Figure 18. Main ANTS API methods used to create a service.

Capsule

The method Execute is the most important because it is called by the Active Node operating system every time a valid capsule arrives at an Active Node. Usually this method will be used by the service developer as the starting point to execute the service business logic. It will include calls to the Active Node API and to the extensions used by the service. The Execute method terminates with a capsule sent to another Active Node or a call to the Active Node operating system method DeliverToApp. A call to the method DeliverToApp signals execution of the service is completed.

Another important method in the Capsule class is the Encode method. This method is used to allow the capsules to send information along the service path inside the Active Network. Its complement method, Decode, receives the information sent.

Application

In the class Application there are two methods used in all services. The method Start is used to invoke the service. This method is responsible for:

- Registering the service with the Active Node.
- Starting the service thread.

The other important method is Receive. The method DeliverToApp invokes the Receive method. The receive method returns results from the execution of the service to the method DeliverToApp and this in turn passes the results back to the Application class.

Protocol

The class Protocol provides the functionality to group the service capsules. The method AddCapsule is used to put a capsule inside a group.

Extension

Extension class method Action is called by the service capsules to start execution of the extension logic.

This design pattern is used to implement all the Active SNMP services discussed in the remainder of this Chapter.

6.3 COMPARISON OF THE STANDARD SNMP SERVICES WITH THE ACTIVE SNMP SERVICES

6.3.1 SPECIFIC SNMP SERVICES

6.3.1.1 SNMP GetBulk Service

Service Definition

This service is typically used to retrieve complete tables from the MIB.

Standard SNMP GetBulk Service

The standard SNMP GetBulk service is achieved by using the SNMP GetBulk Command to retrieve tables stored in the MIB. However, the MIB does not store the size of the tables. The standard GetBulk command merely gives the starting point in the MIB (start of the table for the 1st GetBulk command) and the amount of information to be retrieved. Clearly, it is not possible for SNMP to retrieve the complete table in one command (because it does not store the size of the table).

Active SNMP GetBulk service

There are a number of ways in which the Active SNMP GetBulk service could improve upon the equivalent standard SNMP GetBulk command. This research considered that the retrieval of complete tables was the most useful improvement. This was designed and implemented by locating the upper and lower bounds of the specified table and retrieving the data from the table between these bounds.

Once the Active SNMP GetBulk service is installed as an Active Network service, the procedure to invoke it is initiated when the network manager, using the SNMP Active

Network Toolkit console, selects the GetBulk menu option. As noted in Chapter 5, for the purposes of this research the MIB TcpConnTable table is retrieved. The UML design for this service can be found in Appendix C.1.

Comparison of the standard SNMP GetBulk command with the Active SNMP GetBulk service

The Active SNMP GetBulk service is more efficient than the standard SNMP GetBulk command. In particular, it allows the manager to retrieve any table in the agent MIB using only one request. This contrasts with the GetBulk command in standard SNMP that typically requires many repeat GetBulk commands to be issued by the manager, to retrieve the complete table. Note that the Active GetBulk service could be further improved by allowing, for instance, particular rows in the table to be retrieved. The efficiency of the improved Active Network GetBulk can be compared to the standard SNMP version of the command using a number of metrics. Specifically, the number of messages which must be exchanged between the manager and agent and the delay in retrieving the entire table are fundamental metrics in assessing the performance.

An experiment to compare the time taken to retrieve the TcpConnTable (defined in RFC 1213 [95]) was conducted. In the experiment three different table sizes were investigated. These are referred to as small, medium and large and contain 35, 500 and 1000 rows respectively. Also the distance between the manager console and the agent host was varied such that they were located in the same room, building and campus; these three environments are shown in Figure 19. In practical terms the three environments differ in the distance between the manager and agent and also the number of network devices along the

route between the manager and agent. The experiments were conducted during several weeks at the same time and the arithmetic mean of the results was used to produce the results analysis.

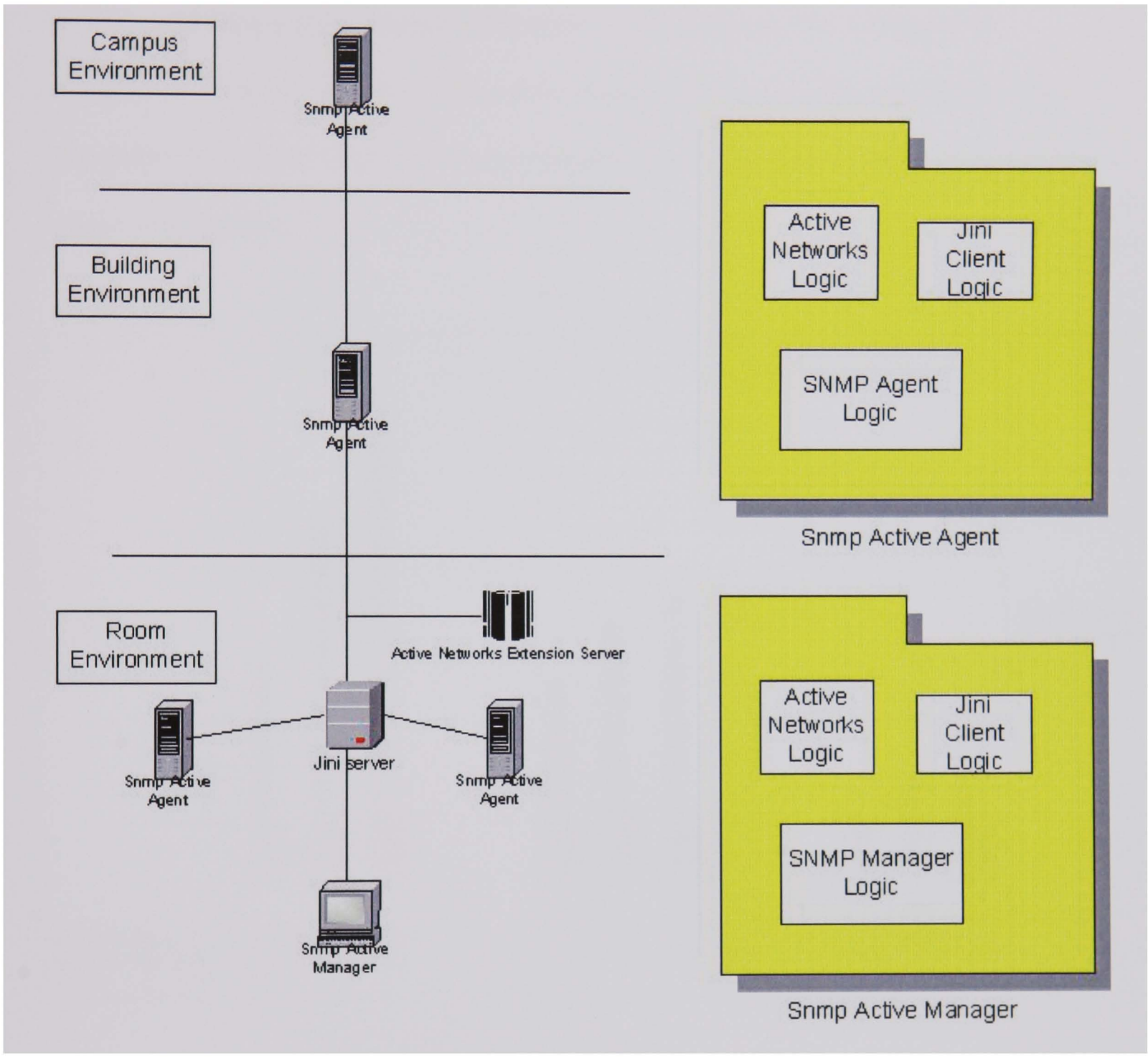
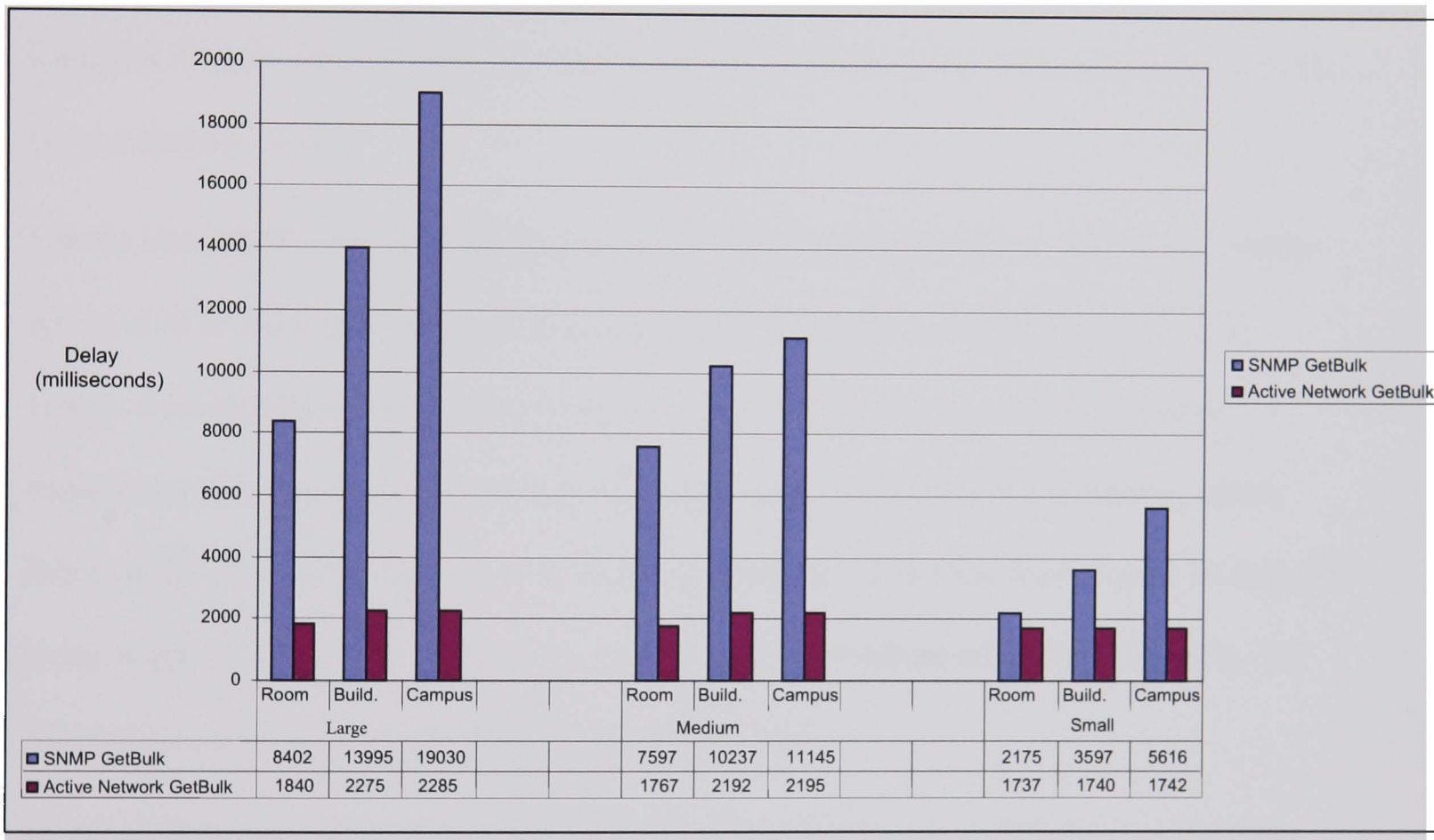


Figure 19. SNMP Active Networks Toolkit

The delay incurred by each GetBulk request includes propagation, processing, queuing and transmission delay. While a detailed discussion of how these various delay measurements

are calculated is beyond the scope of this thesis, nevertheless it is important to note with the exception of transmission delay, each delay factor is a function of the number of packets and not the packet size. It is likely that the standard SNMP version will return more data, and thus will have a higher transmission delay; however this will be negligible in comparison to the overall delay figure and is therefore ignored in this experiment. Hence the greater the number of packets used to transfer a given quantity of data the greater the delay of the transfer.



	Large				Medium				Small		
	Room	Build.	Campus		Room	Build.	Campus		Room	Build.	Campus
SNMP GetBulk	8402	13995	19030		7597	10237	11145		2175	3597	5616
Active Network GetBulk	1840	2275	2285		1767	2192	2195		1737	1740	1742

Figure 20. Delay Performance Analysis for the Active Network GetBulk Service.

Analysis of Results

For the Active SNMP GetBulk service, when the small table is retrieved, the differences among the delays for the 3 environments is less than one percent. [107] This is to be expected because only one GetBulk request is needed to retrieve the complete table and thus the environment will make little difference to the results. A similar comment applies to the retrieval of the medium and large sized tables. Note that there is a medium increase in the delay between the small table and medium table (a twenty per cent for the Campus), and a slight increase between the medium table and large table (a four per cent for the Campus). This increase can be attributed to the extra transmission delay associated with the larger amounts of data.

For the standard SNMP version because of the way in which tables are stored in the MIB, retrieval of an entire table is typically spread across a number of GetBulk commands. Hence, the retrieval of larger tables inevitably requires a greater number of GetBulk requests and thus incurs greater delays. By considering the 'Room' environment, where there are no network devices involved in the data transfer, it is clear from Figure 20 that the delay is greater for the large table size, compared to the medium table size. Similarly, for the medium table size compared to the small table size.

It is also clear from the Figure that for a given table size, there is a difference in the delay incurred by each environment, with the 'Room' environment consistently having the smallest delay and the 'Campus' environment having the largest delay. This is explained by the increasingly larger distances involved and more importantly the number of network

devices deployed for the 3 environments; essentially, each command/response pair needed to retrieve the table, incurs extra propagation, processing and queuing delays.

The results for the standard SNMP GetBulk command do not compare favourably with the Active SNMP Network GetBulk service. In all instances (the three table sizes for each of the three environments) the Active Network version produces significantly smaller delays. Furthermore, as the table size increases, and the environment becomes more complex, the difference in the delay also increases. The choice of table sizes and environments for the experiment are of no real significance other that to produce results for the retrieval of various table sizes across various network environments. In summary, the experiments confirmed that the Active SNMP GetBulk service is more efficient that the standard SNMP GetBulk command, under all conditions, and is more scalable in terms of table size, network size and network complexity.

6.3.1.2 SNMP Threshold Service

Service Definition

A threshold specifies the range of permitted values (minimum and/or maximum) which a MIB variable is expected to operate within.

Thresholds are used frequently in control systems where devices are required to operate within a defined range of values; a temperature gauge for instance. Thresholds are also useful in network management systems and are associated with specific MIB variables. Should the MIB variable be set to a value outside the permitted range then an alarm is raised by the manager. An example would be how the load on a link can be used to select the most appropriate encoding scheme for a call. If the load on a link is within a specified

range then a default encoding scheme is used for a call. If the load drops below the minimum threshold then the call can use an encoding scheme which can effectively increase the amount of bandwidth it uses. Conversely, if the load on the link increases above the maximum threshold then the call must change the encoding scheme so that it uses less bandwidth.

It should be noted that this service can be viewed as a specific application of a more generic monitoring service.

Standard SNMP Threshold Service

There is no SNMP Threshold command. To implement a threshold service the manager must periodically poll the agent to get the current value of the MIB variable. When the current value is returned the manager must test that it is between the minimum and maximum allowed values. If this test fails then the manager will raise an alarm and carry out some appropriate action, depending on the variable. Clearly, many GetRequest/GetResponse command pairs are required. The precise number depends on the length of time that the MIB variable is monitored and the sampling time.

Active SNMP Threshold service

It is a relatively straightforward process to provide an Active SNMP threshold service. The service requires the following information to be defined for a threshold:-

- MIB variable name to be monitored.
- Minimum threshold value.
- Maximum threshold value.

- Default value.
- Delay.

The MIB variable name defines the variable which the threshold is associated with. The user gains access to this service using an option defined in the Active Node menu option. A pop-up allows the user to define the required parameter values which are needed to configure the required service. The delay parameter defines how frequently the MIB variable is checked. The other parameters are self explanatory.

Once the parameter values are defined, the manager can invoke the service. The first action of the service is to send a capsule to the target agent. This capsule will contain the parameter values provided. The capsule will pass these values to its extension class. The extension will create a thread that will run periodically (according to the delay parameter value). For the Active SNMP Threshold service designed and implemented for this research, if the MIB variable goes outside the permitted range, the extension will set the MIB variable to the default value and send a capsule to the manager indicating the occurrence of this event. Clearly, any logic can be programmed for any threshold variables, according to the requirements of the service. Specifically, a maximum threshold was set to Mib-II variable 'tcpCurrEstab' inside the TCP group to know when a maximum number of TCP connections was reached in a node. The minimum threshold was put to zero. When the maximum number or TCP connections in the node passes the maximum threshold, the MIB-II variable 'tcpMaxConn' is set to the default value set. The UML design for this service can be found in Appendix C.2.

Comparison of the standard SNMP Threshold service with the Active SNMP Threshold service

Standard SNMP network management systems cannot efficiently implement threshold variables. A GetRequest command must be sent by the manager, to the agent, requesting the current value of the MIB variable. The agent must respond to that. Thus a GetRequest/GetResponse command pair must be sent periodically to monitor the MIB variable. Furthermore, a 'fat' manager is required to allocate resources to process the responses. Thus, the standard SNMP version of this service is not scalable.

In contrast, the Active Network Threshold service requires exactly one service invocation and one alarm response every time the MIB variable goes outside the specified range. This gives the optimum performance. It was not considered sensible to produce precise quantitative data similar to the GetBulk service. The inherent problem with both these standard SNMP services is the need for multiple GetRequest/GetResponse pairs and thus the Active SNMP service will always out-perform the standard SNMP service. The precise quantitative values will depend on the time the MIB variable is monitored, the sampling period, etc.

A particularly interesting enhancement to this service would be for the manager to delegate to the Agent the intelligence to deal with the event directly, without informing the manager. This takes work from the manager and effectively results in self managing Agents.

The ability to provide this Active SNMP Threshold service and tailor it precisely to the MIB variable which must be monitored, illustrates the flexibility with which new services can be deployed with Active Networks.

6.3.2 Fundamental Operational Network Management Services

SNMP and other network management standards do not have the facility for creating the following three services, thus only the Active SNMP is given below. These services provide solutions to fundamental operational problems inherent in all network management standards.

6.3.2.1 SNMP Macro Network Service

Service Definition

The service allows the agent to execute a Macro by specifying the Macro name. The Macro can be considered to be a script which is composed of SNMP commands or other services.

Active SNMP Macro Network Service

This service allows a manager to execute a Macro with one service invocation. In this research the SNMP MIB Macro Service was used to provide a specific Macro service which executed a number of SNMP GetRequest/SetRequest commands on a number of MIB variables. The objective was to check that the Macro worked, i.e. that all the individual commands were executed, and not necessarily to produce a specific service. Clearly the Macro can contain any number of simple SNMP commands. The Macro service is effectively a design pattern and a developer can use the Macro design pattern to implement any network service. SNMP commands cannot compete with the flexibility, functionality and performance of this Active SNMP service. For the Active version of the Macro service, the strategy to pass the intelligence to the Agents and let the Agent carry out the work independently from the manager achieves the optimum solution.

Specifically, the Macro service was tested by creating a service whose goal was to execute the following SNMP commands:

- To set values in the MIB-II System group variables :
    a. sysContact. The identification of the contact person for the agent node.
    b. sysName. The name for the agent node.
    c. sysLocation. The physical location of the Agent node.
- To get the values for the next MIB-II Ip group variables:
    a. ipInReceives. The variable keeps the total number of input datagrams received, including those received in error.
    b. ipInDelivers. The total number of input datagrams successfully delivered by the Ip protocol.
    c. ipOutRequests. The total number of input datagrams supplied to the Ip protocol as requests for transmission.

This was a simple macro, but nevertheless proved that the Macro service worked. The UML design for this service can be found in Appendix C.3.

6.3.2.2 SNMP Virtual MIB Service

Service Definition

This Dynamic MIB service provides the functionality for the manager to extend the Agent MIB by dynamically creating a new virtual variable.

This value of these virtual variables will typically be computed from other MIB variables. As an example, a virtual variable could be used to store the mean value of another MIB variable, over a specified period of time; this was the service developed for this research. Clearly the complexity of the calculation is dictated by the network manager.

It is interesting to consider the parallels and the synergy which exists between the Macro and the Virtual MIB services. The Virtual MIB service is also a generic service and the classes provided can be extended by the developer to create other virtual MIB variables. The Virtual MIB service therefore can also be considered to be a design pattern. Finally, the benefits of the Virtual MIB service are identical to the Macro service.

When taken together these two generic services provide developer with the functionality to define new virtual variables (data), initialised as appropriate and the functionality to perform any computation (Macro service) on this data. These two services operating together give unlimited power and flexibility to the manager. Specifically, the service was tested by creating a new MIB variable with the mean value of the MIB-II variable 'ipInReceives' during a period of time. The UML design for this service can be found in Appendix C.4.

6.3.2.3 SNMP Manager Delegation Service

Service Definition

This service allows the main manager to delegate management tasks to virtual managers and deploy these virtual managers in specified agents.

To demonstrate the usability of this service, in this research the virtual manager was required simply to get a specified MIB variable value from a specified Agent, over a period of time. However, the Manager Delegation service can be extended to allow the virtual manager to carry out the same tasks as the real manager.

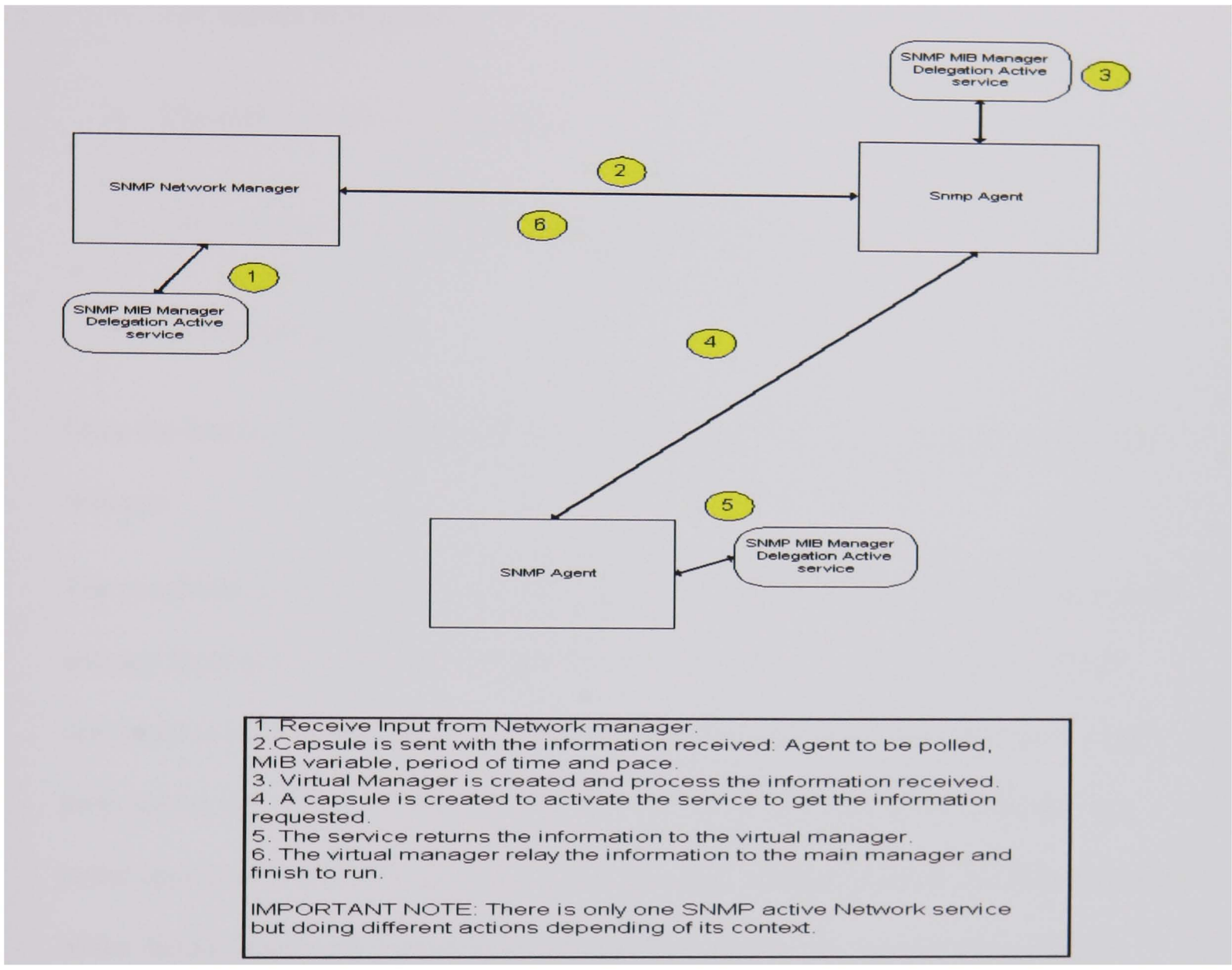


Figure 21. SNMP MIB Manager Delegation Active Network Service.

This flexibility can be used to define different virtual managers, each made responsible for different tasks, as requested by the real manager. Figure 21 shows how this service operates.

The Active SNMP Manager Delegation service requires a number of values to be defined, as follows:-

- The agent where the virtual manager will be placed.
- The agent/s to be polled.
- The MIB variable/s to be polled.
- The period of time over which to sample the variables.
- The sampling period.

Once the data has been collected by the virtual manager it will be forwarded to the main manager.

The possibilities for this service are enormous. Any virtual manager hierarchy can be built and any functionality can be distributed. The virtual managers can execute any SNMP command or make use of the Macro and Dynamic MIB services. There have been other proposals to solve the task delegation problem in SNMP [96] but Active Networks is a better approach as it has the advantage of being easily configured and it can also make use of the Active Macro and Virtual MIB services. Specifically, the service was tested by making the delegated manager to return to the main manager the mean of the MIB-II variable 'ipInReceives' in a specific agent during a period of time. The UML design for this service can be found in Appendix C.5.

6.4 SUMMARY OF THE BENEFITS OF ACTIVE NETWORKS

Having presented the Active SNMP Services and compared these, where appropriate, with the equivalent standard SNMP solution, it is useful t summarise the benefits of Active Networks for network management:

- A significant reduction in network traffic. This is because fewer interactions between the manager and the agents take place.
- A change from a fully centralized paradigm to a higher autonomous paradigm. Using Active Network technology, an Active Network Service can be established as an agent process inside a device. This agent can carry out actions in an autonomous way and its interactions with the manager console can be minimized.
- The possibility of real-time actions. The main problem inhibiting real time operation is network delay. Because Active Network technology distributes intelligence throughout the network, devices are able to solve many problems in real time without involving the manager console.
- It can make the network robust and fault tolerant by giving an efficient and timely answer when a problem arise.
- It can simplify the process of extending and upgrading device agents. The manager console can send an Active Network Service to certain devices to upgrade its system of fault handling for example.
- It can allow the creation of an external view of the MIB data specifically designed for an application. This view can be changed dynamically. For example, an application

may require correlation between several pieces of MIB data. The Active Network approach facilitates this flexibility.

- It can change the network management topology from a hierarchical centralized paradigm to a distributed paradigm in which manager consoles will be distributed and coordinated to solve the network management problems.
- It has the potential to provide a more flexible network management security system. Security can be improved because, firstly, the number of exchanges between the manager console and the agents are minimized and secondly, because it is possible to use several secure transmission techniques in the exchange of messages between the manager console and the agents, depending of the environment and the information interchanged.
- It can be possible to create a 'patrol' and 'first aid' service, which can determine whether the same problem has arisen in a number of devices and treat all of them immediately. The patrol agent effectively acts as a mobile manager system. For example, it could analyze the flow between devices and determine a way of balancing it. It could also correlate several MIB values in different devices, make some calculations and return the results to a manager console for statistical analysis.
- It can make network management systems easier to use and thereby reduce the need for specialized personnel. Work previously carried out by skilled Managers at the central site could now be programmed into active nodes resulting in faster and easier network management.

6.5 SUMMARY

Currently, the most usual task in network management is achieved by having managers poll the managed devices (agents) for data, looking for anomalies. Due to the increase in the number and complexity of nodes in the network, this technique is problematic. In particular, the management centre is inundated with large amount of information, often redundant because there are no changes in the network and frequently, out of date due to the round-trip delay. Active networks can be the natural solution to the 3 operational problems noted above. By making the internal nodes of the network active it is possible to move network management functionality to the 'heart' of the network, reducing delays and bandwidth utilization.

This Chapter has presented a number of Active Networks services to solve a number of already document network management problems. The goal of all of them has been to provide the best technique to every problem and to demonstrate the value of Active Networks as a global solution.

Chapter 7

SUMMARY AND CONCLUSIONS

7.1 INTRODUCTION

This Chapter gives critical review of the research. It summarises the achievements and considers what further work could be undertaken to extend the work, specifically extend the Toolkit. .

7.2 CRITICAL REVIEW OF THE RESEARCH

The main goal of this research was to answer the research question

Can Active Networks solve all the documented problems of Network Management Systems?

To answer this question a number of aims were defined in Chapter 1. How well each of these aims was completed will be reviewed critically in this section.

7.2.1 AIMS OF RESEACH AND CONTRIBUTION TO KNOWLEDGE

7.2.1.1 To produce a thorough analysis of the strengths and weaknesses of SNMP

Exhaustive research was conducted, using the main journals and sources where information about network management issues and SNMP are typically published, together with books and other literature. An internal report was produced [104] and the contents were used in

Chapter 2 to present the problems of SNMP. Since the publication of this report no further weakness have been identified.

7.2.1.2 To construct a framework which will allow a comparison to be made between standard SNMP services and Active SNMP services

A test-bed was needed to compare standard SNMP network management services (commands) with the corresponding Active SNMP network management services. A framework was designed and using this framework the test-bed was built. The test-bed comprises a complete standard SNMP network management system and this was augmented to produce a separate Active SNMP network management system. Note that the Active SNMP version defaults to a standard SNMP system and that it is a platform only, in that at this stage no services have been designed and built. The design of the test-bed was presented in Chapter 5.

When services are created and deployed in this test-bed it will then be possible to compare them.

7.2.1.3 To design and build a set of services which allow a comparison to be made between standard SNMP services and Active SNMP services

It is accepted that SNMP has many problems. Some problems are specific problems, for instance the GetBulk command is very inefficient. Other problems are more major and are central to the difficulties SNMP (and other network management standards) has with responding to the changing needs of networks in general, for instance new technologies. In Chapter 3, 2 specific problems and 3 major operational problems with SNMP were chosen

as being crucial to answer the research question. Services were designed and built which provided solution to these problems and the details are given in Chapter 6.

7.2.1.4 To provide an ANTS system to dynamically load extensions in the Active Nodes

While ANTS provides the functionality to dynamically load capsules, it did not provide the equivalent functionality for the Extension code. This was considered a weakness of ANTS. Therefore a system to dynamically load the Extension code was designed and built as part of this research. This system works correctly and was used in the execution of the Active SNMP services.

7.2.1.5 To provide an ANTS system to dynamically load Active Network topology

ANTS uses a configuration file to establish the Active Network topology. While this may be acceptable in a research environment it is not considered acceptable in a commercial environment. Therefore a system was designed and built which dynamically maintains the network topology; the design of this system was discussed in Chapter 5.

The system works correctly and was used in the execution of the Active SNMP services.

7.3 RESEARCH QUESTION

The question the research set out to answer was:-

Can Active Networks solve all the documented problems of Network Management Systems?

The services designed and built to answer this research question were classified into specific SNMP problems, and more general network management operational problems. The specific SNMP problems relate to the inefficient way in which SNMP executes certain commands, while the operational problems relate more to the inherent way in which network management systems were designed.

The results are conclusive.

The Active SNMP services for the specific SNMP problems have been shown to be much more efficient compared to the standard SNMP version. In particular, the use of network bandwidth was optimal and the delays (response times) were much smaller.

Three Active services were built to address the operational problems of network management systems (it should be noted that no facility exists in any network management standard to provide these services):-

- SNMP Macro Network Service – effectively allows a script be executed, where the script is a set of basic commands or other services. This solves the polling problems and the inherent efficiencies of this.

- SNMP Virtual MIB Service – allows new MIB variables to be defined, on demand. This also allows 'views' of the MIB to be defined and thus views can be tailored to specific services. To define a virtual MIB variable the existing MIB does not need to be altered. Therefore this service provides a means for integrating new technologies and the MIB variables required to support technologies without the need for proprietary solutions.

- SNMP Manager Delegation Service – this allows responsibility to be delegated to agents. In effect agents can be viewed as virtual managers. In this way the management function can be distributed throughout the network, to any agent. This solves the problems caused by the centralised paradigm, including the scalability problem.

Each of these services provide solutions to major operational problems; these problems have been discussed frequently in the literature [17,20,36], etc. However, the power comes from their synergy. The Toolkit built as part of this research provides the ability to define any variable (view) in the MIB, on demand AND perform any computation on any MIB variable AND distributed this computation anywhere in the network. I believe therefore that these three services form the generic solution to all the general network management problems set out in Chapter 1. Furthermore this research allows legacy systems to operate alongside newer technologies and allows services to be deployed incrementally.

Therefore the answer to the research question is YES.

7.4 RELATED WORK

'The Smart Packets Project' [54], developed in BBN Systems and Technologies, is a complete Active Network Architecture designed specifically to provide an Active network management system. Thus this project is not a generic Active Networks technology. Smart Packets does not address the major operational problems which exist in network management standards, but instead is a replacement for current network management

standards. Therefore no consideration is given to legacy systems and thus its usefulness is limited.

A research project complementary to this research is being developed at the Swiss Federal Institute of Technology. [98] The project focus is the management of an Active Network. In an Active Network there is a need to debug services injected into the network. The project proposes an Active Network Architecture which will be used as prototype for exploring issues of resource management and network management inside an Active Network.

Finally, the use of intelligence inside the network was suggested by G. S. Goldszmidt. [6] One of the focus areas in this project was network management as means to prove the use of mobile agents inside the network. The research brings in the use of mobile agents for remote execution. It introduces a language-independent agent technology. Its delegated agents are dispatched to remote elastic processes and are dynamically linked with them and executed as threads under remote control. The project uses this architecture to manage networked systems.

Active Networks also puts intelligence into the network through services and therefore can provide the same functionality as agents.

7.5 FURTHER WORK

7.5.1 CATER FOR ALL NETWORK MANAGEMENT STANDARDS

To make the toolkit more useful it is proposed that it caters for all the network management standards, for instance CMIP and TNM. Active Networks operates by injecting intelligence into the network, transparently. Thus Active Networks is independent of the network

infrastructure and thus the network management standard. Therefore to be able to accommodate other network management standards, only the basic commands (e.g. the equivalent of the GetRequest command) need be implemented. The result will be a Toolkit which can operate with any device, regardless of the network management standard deployed in the device.

7.5.2 Open Source experimental

The Toolkit could be made available as Open Source and used in much the same way as Linux. In particular, developers could compose new services (using the Macro service) and make these available, publicly. This Toolkit could then be used as the basis for network management research and testing. The benefits of this are:-

- Services can be made available immediately as there is no need for the long standardisation process – developers will select from the services provided.
- The automatic Active Network deployment service (improved by this research) can be used to deploy new versions of the services.

This Toolkit could be complemented with an Active Directory system where all the available Active Network services are registered. The use of this system will improve the security inside the Active Network by controlling which services will be provided to the Active Networks community. Jini was used to provide a similar service as part of this research and thus is a candidate technology to provide this service.

7.5.3 USE OF XML

The XML language standard could be used as the basis for communication between the Active Network services. It could also be use as the language for writing the Macros.

7.5.4 ACTIVE NETWORKS SYSTEM SERVICES

Active Networks can be use to provide system services, for instance, resource management services. In this Active Network can be a self-managing system.

7.6 FINAL CONCLUSIONS

This research has shown that Active Networks can solve even major operational problems in network management systems. It has produced a Toolkit which provides the design patterns to execute any Macro define any new MIB variable and automatically distributed these to any device in the network. The further work has argued that this could be made Open Source and used by developers as a test-bed for experimental work on network management systems.

Bibliography

[1] Marshall T. R. *The Simple Book. An introduction to networking management. Revised second edition.* Prentice Hall Series in Innovative Technology.1996.

[2] Tennenhouse, D.L. *A Survey of Active Network Research.*IEEE Communications Magazine,Vol 35,No 1, pp 80-86. 1997.

[3] Black U. *Network Management Standards. SNMP, CMIP, TMN, MIBs and Object Libraries. Second Edition.* McGraw-Hill, Inc.1995.

[4] Terplan K. *Communication Networks Management.* Englewood Cliffs, NJ:Prentice-Hall 1992.

[5] Stallings W. *SNMP,SNMPv2, and RMON: practical network management – Second Edition.* Addison-Wesley Publishing Group. One Jacob Way. Reading. Massachusetts. USA. 1997.

[6] Goldzmidt G. S. *Distributed Management by Delegation.* Proceedings of the 15th International Conference on Distributed Computing Systems, IEEE Computer Society.1995.

[7] Mier, E. *Evaluate SNMP on Bridges.* Network World. 1991.

[8] Mier,E. *Test Routers SNMP Agents.* Network World.1991.

[9] Masum Z. Hasan. *The Management of Data,Events, and Information Presentation for Network Management.* Thesis submitted at University of Waterloo,Ontario,Canada.1996.

[10] Tennenhouse D.L, Wetherall D. J. *Towards an Active Network Architecture.* In Multimedia Computing and Networking (MMCN'96), San Jose, CA.1996.

[11] Wetherall, D.J. *Service introduction in an Active Network.* Thesis submitted at Massachusetts Institute of Technology. 1999.

[12] Miller M. A.*Managing Internetworks with SNMP.* IDG Books Worldwide, Inc.1997.

[13] Satz G.L. *A New View on Bulk Retrieval with SNMP.* The Simple Times. Volume 1 Number 1. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1992.

[14] Sprenkels R. *Bulk Transfers of Mib Data.* The Simple Times. Volume 7 Number 1. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1999.

[15] Yoshihara K.,Sugiyama K.,Horiuchi H.,Obana S. *Dynamic polling algorithm based on network management information values.* IEICE Transactions on Communications. 1999.

[16] Cheikhrouhou M.,Labetoulle J. *A efficient polling layer for SNMP.* NOMS 2000. 2000 IEEE/IFIP Network Operations and Management Symposium. IEEE, Piscataway, NJ,USA. 2000.

[17] Hwang K. C., Hong J.J., Lee K. H. *A SNMP group polling for the management traffic.* Proceedings of IEEE. IEE Region 10 Conference. TENCON 99. Multimedia Technology for Asia-Pacific Information Infrastructure. IEEE,Piscataway,NJ,USA.1999.

[18] McClenaghan D., Neisser G. *Manchester University Tiny Network Element Monitor (MUTiny NEM) - a Network/Systems Management Tool.* Terena Networking Conference.2002.

[19] Ochiai T. *Network Management using Distributed Computing.* IEICE Transactions on Communications. 1995.

[20] Gavalas D.,Greenwood D.,Ghanbari M., O'Mahony M. *An infrastructure for distributed and dynamic network management based on mobile agent technology.* IEEE International Conference on Communications.1999.

[21] Kornegay M.L. *Toward Useful – and Standardized- SNMP Management Applications.* The Simple Times. Volume 2 Number 2. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1996.

[22] Wadbusser S. *The Trend Towards Hierarchical Network Management.* The Simple Times. Volume 2 Number 6. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1996.

[23] Leri D. *Introduction to the Script MIB.* The Simple Times. Volume 7 Number 2. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1999.

[24] Guo Y.,Hiranaka Y., Akatsuka T. *A simple hierarchical network management system based on extended SNMP.*Joint Conference of Intelligent Systems (JCIS'98). Assoc. for intell. Machinery, USA. 1998.

[25] Stallings W. *SNMPv3: A Security Enhancement for SNMP.* IEEE Communications Surveys. 1998.

[26] Breitbart Y.,Garofalakis M., Martin C.,Rastogi R.,Seshadri S.,Silberschatz A. *Topology discovery in heterogeneous IP networks.* Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies,NJ,USA. 2000.

[27] Waldbusser S. *Exposing the Myths about Autotopology.* The Simple Times. Volume 1 Number 1. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1992.

[28] Rose M. T. *Industry Comment.* The Simple Times. Volume 1 Number 3. c/o Dover Beach Consulting, Inc. 420

Whisman Court, Mountain View California. USA. 1992.

[29] Lixia Zhang, Scott M., Floyd S., Jacobson V., Nguyen K., Rosenstein *A. Adaptive Web Caching: Towards a New Global Caching Architecture.* 3rd International WWW. Caching Workshop. Manchester, England, 1998.

[30] Wellens C. *Towards Useful Management.* The Simple Times. Volume 4 Number 3. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1996.

[31] Waldbusser S. L. *Should user interface information be standardized.* The Simple Times. Volume 2 Number 1. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1993.

[32] Steward R. L. *Working Group Synopsis.* The Simple Times. Volume 4 Number 2. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1992.

[33] Baker F. J., Kostick D. C.,Tesink K. *Working Group Synopsis.* The Simple Times. Volume 2 Number 5. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1993.

[34] John A., Vanderveen K.,Sugla B. *A Java-based SNMP agent for dynamic MIBs.* Global Telecommunications Conference. GLOBECOM'99. IEEE, Piscataway, NJ, USA. 1999.

[35] Natale B. *Introduction to Agent Extensibility.* The Simple Times. Volume 4 Number 2. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA.1996.

[36] Chauki J.,Shahsavari MM. *Component-based distributed network management.* Proceedings of the IEEE SoutheastCon 2000. IEEE, Piscataway,NJ, USA. 2000.

[37] Gavalas D.,Greenwood D.,Ghanbari M.,O'Mahony M. *Complimentary polling modes for network performance management employing mobile agents.* Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. IEEE, Piscataway,NJ, USA. 1999.

[38] Zapf M. Herrmann K. Geihs K. *Decentralized SNMP management with mobile agents.* Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management. IEEE, Piscataway, NJ, USA. 1999.

[39] Gavalas D.,Greenwood D.,Ghanbari M.,O'Mahony M. *Intelligent Agents for Telecommunication Applications.* Third International Workshop, IATA'99. Proceedings,Springer-Verlag, Berlin,Germany. 1999.

[40] Koch FL. *Autonomous agents for computer network management.* PAAM

98. Proceedings of the Third International Conference on the Practical Application of Intelligent agents and Multi-Agent Technology. Practical Application Co. Ltd, Blackpool, UK. 1998.

[41] Goncalves-Rubinstein M,Bandeira-Duarte O.C. *Evaluating the performance of mobile agents in network management.* Global Telecommunications Conference. GLOBECOM'99. IEEE, Piscataway, NJ, USA. P.386-390. 1999.

[42] Kolaczek G. *The advantages of distributed management.* Informations Systems' Architecture and Technology. ISAT'97. Proceedings of the 19th ISAT International Scientific School. Oficyna Wydawnicza Politech. Wroclawskiej, Wroclaw, Poland. 1997.

[43] Pras A. *Editorial.* The Simple Times. Volume 7 Number 2. c/o Dover Beach Consulting, Inc. 420 Whisman Court, Mountain View California. USA. 1999.

[44] Goldszmith G., Yemini Y. *Computing MIB Views via Delegated Agents.* Proceedings of the IEEE Third International Workshop on Systems Management. IEEE Comput. Soc, Los Alamitos, CA, USA. 1998.

[45] Thottan M.,Chuanyi Ji. *Proactive anomaly detection using distributed intelligent agents.* IEEE Network. 1998.

[46] Baras JS.,Ball M.,Gupta S.,Viswanathan P.,Shah P. *Automated network fault management.* MILCOM 97 Proceedings. IEEE, New York, NY, USA. 1997.

[47] Duarte EP Jr.,Nanya T. *An SNMP-based implementation of the adaptive distributed system-level diagnosis algorithm for LAN fault management.* NOMS 96. IEEE Network Operations and Management Symposium. New York, NY. USA. 1997.

[48] Kaufman D. *Six common myths about the simple network management protocol.* EDPACS. 1999.

[49] Kato N.,Ohta K., Ika T.,Mansfield G., Nemoto Y. *A proposal of event correlation for distributed network fault management and its evaluation.* IEICE Transactions on Communications. 1999.

[50] Chiu T. *Getting proactive network management from reactive network management tools.* International Journal of Network Management. 1998.

[51] Bhattacharjee S., Calvert K. L., Zequra E.W. *An Architecture for Active Networking.* Proc IEEE INFOCOM'97. 1997.

[52] Decasper D. Plattner B. DAN: *Distributed Code Cashing for Active Networks.* Proc. IEEE INFOCOM '98, San Francisco, CA. 1998.

[53] Wetherall D. J.,Tennenhouse D.L. *The ACTIVE-IP option.* Proceedings of the the

7th ACM SIGOPS European Workshop. 1996.

[54] Schwartz B.,Zhou W., Jackson A. W. *Smart Packets for Active Networks.* BBN Technologies. 1988.

[55] Banchs A., Effelsberg W., Tschudin C., Turau V. *Multicasting Multimedia Streams with Active Networks.* Proc. IEEE Local Computer Network Conference LCN'98, Boston, MA.1998.

[56] Gunter C.A., Nettles S. M., Smith J. M. *The SwitchWare Active Network Architecture.* IEEE Network. Special issue on Active and Programmable Networks.1998.

[57] Yemini Y., Silva S. *Towards Programmable Networks.* Proc. IFIP/IEEE int'l Workshop on Distributed Systems, Operations, and Management, L'Aquila, Italy. 1996.

[58] Bhattacharjee S., Calvert K. L., Zegura E. W. *On Active Networking and Congestion.* Technical Report GIT-CC-96/02.1996.

[59] Bhattacharyya S., Towsley D., Kurose J. The Loss Path *Multiplicity Problem in Multicast Congestion Control.* UMass CMPSCI Technical Report TR 98-76. 1998.

[60] Levine B.N., Garcia-Luna-Aceves J.J.. Improving Internet Multicast with Routing Labels. Proc. IEEE ICNP '97, Atlanta, Georgia. 1997.

[61] Clark D., Shenker S. Zhang L. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.* Proc. ACM Sigcomm '92'. 1992.

[62] Lehman L. H., Garland S.J., Tennenhouse D.L. *Active Reliable Multicast.* Proc. IEEE INFOCOM '98. San Francisco, CA.1998.

[63] Bhattacharjee S., Calvert K. L., Zegura E. W. *Self Organizing Wide-Area Network Caches.* Proc. IEEE INFOCOM '98, San Francisco, CA. 1998.

[64] Floyd, S. *Adaptive Web Caching.* Boulder Cache Workshop '97, 1997.

[65] Partridge, C., Strayer W. T., Schwartz B. I., Jackson A. W.*Commentaries on Active Networking and End-to-End Arguments.* IEEE Network.1998

[66] Goldzmidt G. S. *Distributed Management by Delegation.* Proceedings of the 15th International Conference on Distributed Computing Systems, IEEE Computer Society. 1995.

[67] Prashant C. *Darwin: Customizable Resource Management for Value-Added Network Services.* Proc. Sixth IEEE int'l Conf. On Network Protocols (ICNP'98), Austin. 1998.

[68] Tschudin C.,Lundgren H.,Gulbrandsen H. *Active Routing for Ad Hoc Networks*.IEEE Communications Magazine. 2000.

[69] Parveen P. *An introduction to active network node operating systems.* Crossroads, v.9 n.2, p.21-26.2002

[70] Qie X., Bavier A., Peterson L., Karlin S. *Scheduling computations on a software-based router.* ACM SIGMETRICS Performance Evaluation Review. 2001.

[71] Chen W. E., Hu C. L. *A mobile agent-based active network architecture for intelligent network control.* Informatics and Computer Science: An International Journal. 2002.

[72] Ott D. E., Mayer-Patel K. *Coordinated multi-streaming for 3D tele-immersion.* Proceedings of the 12th annual ACM international conference on Multimedia, New York, NY, USA.2004.

[73] Calvert K. L., Griffioen J., Wen S. *Lightweight network support for scalable end-to-end services*. Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications. Pittsburgh, Pennsylvania, USA. 2002.

[74] Stoica I., Adkins D., Zhuang Shelley., Shenker Scott., Surana Sonesh., *Internet indirection infrastructure*. ACM SIGCOMM Computer Communication Review. 2002.

[75] Grimm R., Bershad B. N., *Separating access control policy, enforcement, and functionality in extensible systems*. ACM Transactions on Computer Systems (TOCS). 2001.

[76] Calder M., Kolberg Mario., Magill E. H., Reiff-Marganiec Stephan. *Feature interaction: a critical review and considered forecast*. Computer Networks: The International Journal of Computer and Telecommunications Networking. 2003.

[77] Kummerfeld F.A., Fekete B., Hitchens K., Basser M. *A new dynamic architecture for an active network.* Open Architectures and Network Programming, 2000. Proceeding. OPENARCH 2000. 2000.

[78] Tullmann P., Hibler M., Lepreau J. *Janos: A Java-Oriented OS for Active Network Nodes*. 2002 DARPA Active Networks Conference and Exposition (DANCE'02) San Francisco, CA. 2002.

[79] Bernadat P., Feeney L., Lambright D., Travostino F. *Java sandboxes meet service guarantees: Secure partitioning of CPU and memory.* Technical Report TOGRI-TR9805, The Open Group Research Institute. 1998.

[80] R.L. Rivest. *The MD5 Message Digest Algorithm*. RFC 1321.1992

[81] Sun MicroSystems Inc. *Java Management Extensions White Paper.* Revision 0.1. 901 San Antonio Road. Palo Alto, California. 94303-4900. USA.1999.

[82] Sun MicroSystems Inc. *Java Dynamic Management Kit White Paper.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA. 2000.

[83] Sun MicroSystems. *Jini Technology Executive Overview. Revision 1.0.*901 San Antonio Road. Palo Alto, California. 94303-4900. ,January 1999.

[84] Max G. *Java in the management sphere.* Technical article. *Java World Magazine.* 501 Second St. San Francisco, CA 94107. USA. 1999

[85] Sun MicroSystems Inc. *Java Management Extensions Instrumentation and Agent Specification, v1.0.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA. 1999

[86] Sun MicroSystems Inc. *Java Management Extensions SNMP Manager API.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA. 1999.

[87] *Abstract Syntax Notation One (ASN.1)* .Specification of Basic Notation ITU-T Rec. X.680 (2002) | ISO/IEC 8824-1. 2002

[88] Sun MicroSystems Inc. *Jini Architecture Specification.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA.1999.

[89] Carriero N., Gelernter D. *The S/NET's Linda Kernel.* Technical Report 383, Yale University Department of Computer Science.1985.

[90] Gong L, *Java Security Architecture(JDK1.2).* 1998.

[91] Sun MicroSystems Inc. *Jini Distributed Events Specification.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA.1999.

[92] Sun MicroSystems Inc. *Jini Distributed Leasing Specification.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA.1999.

[93] Sun MicroSystems Inc. *Jini Transaction Specification.* 901 San Antonio Road. Palo Alto, California. 94303-4900. USA.1999.

[94] Contributing Members of the UPnP™ Forum. *UPnP™ Device Architecture. Version 1.0.*2000.

[95] McCloghrie K., Rose M.

RFC 1213 - Management Information Base for Network Management of TCP/IP-based internets:MIB-II. 1991.

[96] Heintz L.,Gudur S. *RFC 2742. Definitions of Managed Objects for Extensible SNMP Agents.* 2000.

[97] Alexander D.S., Braden B., Gunter C.A.,Jackson W.A., Keromytis A.D., Milden G.A., and Wetherall D.A. *Active Network Encapsulation Protocol (ANEP).* Active Networks Group Draft. 1997.

[98] Plattner B., Brunner M., *Management of Active Networks*. ICC Workshop on Active Networking and Programmable Networks, Atlanta, 1998.

[99] J Case, M Fedor, M Schoffstall, J Davin. *A Simple network management protocol (SNMP) The Working Group, Internet.* Engineering Task Force. RFC 1157,1990.

[100] OSI, I.S.O. *Systems Management Overview*, 1991.

[101] OSI,ISO. *InformationTechnology, Open Systems Interconnection,Common Management Information Services Definitions*, 1991.

[102] ITU-T Recommendation M.3000 (02/00). *Telecommunications management network. Overview of TMN Recommendations*.2000

[103] ITU-T Recommendation M.3010 (02/00).*Telecommunications management network. Principles for a telecommunications management network.* 2000

[104] A.C. Rivas. 'Analysis of the Simple Network Management Protocol'. De Montfort University. Communication Networks Research Group Technical Report 01-2001.UK. 2001.

[105] A.C.Rivas, A. Platt, M. Morse. 'Jini. The future of Distributed Object Management'. 7th European Concurrent Engineering Conference. UK.2000.

[106] A.C.Rivas, A. Platt..'Network Management Using Active Networks'.SofCOM 2001. Proceedings of the International Conference On Software,Telecommunications & Computer Networks. Italy.2001.

[107] A.C. Rivas, A. Platt. 'Active Networks as a solution to Network Management'. ICCC 2004. Proceedings of the International Conference on Computers and Communications. Romania.2004.

Appendix A

THE ANTS API

The description of the ANTS architecture schematizes its main components and its functions but has left without explaining the details of its use and implementation.

It was mentioned before that the ANTS toolkit has been written using the language Java entirely. This section tries to give more information about the key classes used by the toolkit to create an active network. It is important to indicate that the node classes and their local applications are executed in a single Java Virtual Machine using different threads.

OVERVIEW OF MAIN CLASSES

The ANTS architecture defines the active nodes as elements of the network that are connected by channels, and with capsules like agents that are injected inside the network by applications. In the ANTS toolkit these concepts are matched to concrete java classes.

Each active node is represented by an object of the class Node. The functionality of a node can be extended using instances of subclasses of the abstract class Extension. Each network interface is represented by an object of the class Channel. Different channels types (point to point or shared medium) could be used when it is necessary.

New applications of a service are developed creating subclasses of the abstract class Application. The class that represents the service is the abstract class Protocol. The capsules are created creating subclasses of the classes Capsule. The new capsules are organized in Code Groups using methods of the Protocol class.

THE NODE CLASS

The class node represents the core of an Active Node. The class exports a set of services to other classes in the toolkit: registration of protocols and extensions, control of capsules execution, etc. These services can be divided in four categories.

The first category has as common element the return of information about the active node environment. Functions as GetAddress and Time return the local node Internet address and the local time respectively. The method GetChannel allows obtaining the Channel object used for the forwarding of the capsule. On the other hand, the method FindExtension allows finding the instance of an extension installed in the node.

The following category of functions is related to the Node storage manipulation system. Per example, the method Put allows setting an object inside the store after decrementing the capsule's resources limit. On the other hand, the method Get allows recovering the object meanwhile the method Remove allows to eliminate it of the system.

Another category is formed by the functions to control the capsules processing flow and its propagation to other nodes of the network. The method RouteForNode allows forwarding a capsule toward a node using the default routing system, decrementing the capsule's resources limit in the process. The method DeliverToApp allows invoking an instance of local application to deliver a capsule.

The rest of the functions form a category whose main function is allowing the registration of the service main elements in the node structure. Methods as RegisterProtocol, RegisterApplication and RegisterExtension permit respectively to

register an instance of the classes Protocol, Application and Extension in the node are a clear example.

THE CAPSULE CLASS

Subclasses of the class Capsule are used to define the PDUs that are used in a service. Inside the fields of a capsule, the source and destination addresses are defined as 32 bit integers, with the auxiliary class NodeAddress providing routines to format the addresses according to the notation used in the protocol IP. The class provides a number of methods to obtain the information set in the capsule fields:

- GetResource which returns the value of the resources limit field.
- GetPrevious is used to obtain the address of the node previously visited by the capsule.
- GetCapsuleID, GetGroupID and GetProtocolID are used to obtain the capsule, code group and protocol fingerprints respectively.

ANTS provides a convenient class that the programmers of services can use in place that Capsule. Its name is DataCapsule and is specially designed to work with the protocol UDP.

One of the programmer's of services obligations is to override the methods used in the process of a capsule. The reason for this obligation is because the normal process of a node depends on these methods for the manipulation of the capsules.

For example, the node presupposes that the logic of a capsule is inside the method Evaluate. This method is invoked every time that a capsule arrives to a node, allowing

to the capsule to obtain an object reference of the class node. The method should manage the errors that can arise in its execution by means of catching exceptions, and it should finish its process in the time allowed by the node. While the method is executed, the node operating system guarantees that the execution is not affected by the execution of other capsules. The only actions that the can carried out, besides calling to the methods of the API of the node, it is to call to other capsules of the same code group or to communicate with other capsules of the same code group.

Finally, new capsules generally use class variables to represent additional fields in the PDU to transport custom data among different nodes. This will permit to transport information about events in the nodes visited by a capsule. The methods Encode, Decode and Length allow converting these fields to and from an external representation of the data to avoid data representation problems. The external representation is provided by the auxiliary class Xdr.

THE PROTOCOL CLASS

To define a new service, besides creating capsules and extensions, the programmer of services should create a class that extends the class protocol. This class is used to organize the capsule inside code groups. It is obligatory that the class provides with a constructor method without parameters where the organization of the capsules is exposed. The methods StartGroupDefn and CloseGroupDefn are used for the creation of a code group while the method AddCapsule is used to add a capsule to the code group. The Figure 22 shows an example of the use of these methods in the creation of a protocol element.

```
public class SnmpDynamicMibProtocol extends Protocol {

        public SnmpDynamicMibProtocol() throws Exception {
                startProtocolDefn();
                startGroupDefn();
                addCapsule("uk.ac.dmu.activenetworks.apps.SnmpDynamicMibCapsule");
                endGroupDefn();
                endProtocolDefn();
        }
}
```

Figure 22. Protocol class example.

THE APPLICATION CLASS

An application is an independent entity that makes use of the services that ANTS provides. It is built by means of the extension of the class Application. The class provides a series of methods that allow making use of the local node, to register protocols, inject capsules in the network and to receive capsules from the network.

One of the first obligations of the class application in a service is to create an object of the corresponding protocol and to register it with the local node using the method Register that the API of the node provides. This step allows to the local node to obtain the code of the capsules that form the protocol from the local files system to be able to initialize the code distribution system. Once completed the registration, the application could send and to receive capsules of the service. The method Send is used to send a capsule to the network. The answer from the network is received by means of the method Receive. This method is called by the node when a capsule coming from the network has as destination the local application.

THE EXTENSION CLASS

The class Extension is the mechanism used by application services to increase the functions that a node can offer. New extensions are developed by means of the extension of the class Extension. Once created the extension, it is necessary to install it like part of the node. The method AttachExtension of the class Node allows the installation of the extension in the node operating system. Because of the node security infrastructure, an extension should be accessed by means of the method FindExtension of the class node. This method allows to a capsule to look for an extension installed in the node, using the name of the extension as key. The method returns a reference to the extension or null in the event of not finding the extension. Once a capsule has the extension instance it can use the public methods that provides.

Appendix B

THE JMX/JDMK API

In JMX there are two main packages (Java libraries) which are used to create an SNMP system. [86] The first of these, javax.management.snmp, contains:

- The classes and interfaces needed to deal with SNMP (versions 1 and 2) protocol data.
- The Java classes used to make and process SNMP requests and responses.
- The Java classes needed to build a prototype SNMP MIB.

This package is used to build both agents and managers.

The second main package, javax.management.snmp.manager contains a set of interfaces and classes needed to create an SNMP manager. Since an understanding of the structure and use of these packages is crucial when building a new, Java-based, network management system, they are described in detail in the following paragraphs.

The JDMK API is one particular implementation of the JMX API and provides the developer with the means to build a SNMP network management system. Note that systems based on other network management standards can be build using the JDMK API. There are 3 packages in the JDMK API relating to SNMP. These are com.sun.jdmk.snmp,com.sun.jdmk.snmp.agent and com.sun.snmp.IPAcl.

JMX PACKAGES

PACKAGE JAVAX.MANAGEMENT.SNMP

There are four interface definitions in this package. The first two of these, SnmpDataTypesEnums and SnmpDefinitions, simply identify the constants used in generating and processing request and response messages which pass between entities in an SNMP system.

Interface SnmpOidTable specifies how a MIB must be implemented in a SNMP Manager. The MIB defines the variables which are to be managed and these are structured as a tree. Each variable has a unique logical name, which comprises a set of identifiers separated by periods. The individual identifiers of the name refer to successive branches of the tree. Similarly, variables also have equivalent unique numeric identifiers, which comprise a set of numbers separated by periods; these are known object identifiers (Oids). Implementations of the interface SnmpOidTable create an array of SnmpOidRecords, one SnmpOidRecord for each MIB variable required. Each SnmpOidRecord contains a copy of the reference information for that particular MIB variable as defined in the standard (description, status, etc), for that particular MIB variable. The MIB is used by the SNMP Manager to retrieve the reference information for a given MIB variable name or equivalent Oid. Thus the manager MIB effectively provides little more than a dictionary service. Note that the SnmpOidTable interface can also be used to define a MIB in the agent. However, because the agent MIB must also store the current MIB variable value (in addition to all the reference information) the agent MIB is typically defined in another way; this will be described later.

The fourth and final interface in the package is SnmpPduFactory. This interface defines the method EncodePDU that encodes a given SnmpPduPacket object and returns the resulting SnmpMessage object. The method DecodePDU does the reverse. The class SnmpPduFactoryBer is the default implementation of this interface. It encodes the information using the BER (Basic Encoding Rules) standardised encoding scheme associated with ASN.1[87], but clearly other encoding schemes could also be implemented. The UsePduFactory method of the SNMPAdaptorServer provides the means for a SNMP agent to select an encoding scheme from any implementations of this interface. A similar facility is provided for the SNMP manager. In particular, a SnmpPeer object, which represents an agent in the manager, defines a SetPduFactory method. A call on this method associates an implementation of SnmpPduFactory (essentially an encoding scheme) with the agent object. This facility gives the manager the flexibility to associate different encoding schemes with different agents. Note that SnmpPeer also provides a GetPduFactory method which returns details of the SnmpPduFactory object associated with the agent.

In addition to the four interfaces, there are a number of classes defined in the javax.management.snmp package. Many of these classes simply provide support for the low level manipulation of the various types of data contained within the PDUs and as such add little to the understanding of the JMX SNMP API. However there are a number of classes in the package which warrant closer scrutiny.

The Class SnmpPduPacket, as its name indicates, represents a generic SNMP Pdu and defines all the individual fields associated with the various SNMP Pdu types. Of particular interest is the field which carries the actual data sent / received between the manager and agent. This field is represented by the SnmpVarBindList class which

simply defines an array of SnmpVarBind objects. Each SnmpVarBind object contains a SnmpOid object and an associated SnmpValue object. The SnmpOid object stores the unique numeric identifier of the SNMP Mib variable as an array of long numbers, one for each number in the identifier and the SnmpValue object stores the current value of the variable. Thus the SnmpVarBindList is simply a list of MIB variable identifiers together with their corresponding values. Note that the class SnmpOid does not store the name of the MIB variable; however, it is possible to instantiate a SnmpOid object use the name, in which case the SnmpOidTable is used to retrieve the corresponding numeric identifier. Clearly, the SnmpOidTable object must exist before any PDUs can be exchanged.

Note also that SnmpValue is an abstract class and thus the type of an object of this class cannot be obtained directly. Instead the GetType method of the corresponding SnmpRecord object (MIB variable record) is used. To do this the SnmpOidRecord object must first retrieved from the SnmpOidTable object (MIB) using the GetSnmpOidTable method of the SnmpOid object.

Class SnmpPduPackets has a number of subclasses, each of which represents one or more specific SNMP Pdu types. For instance, the class SnmpPduBulk represents the SNMP getBulk Pdu type and the class SnmpPduRequest represents the SNMP get, get-next, set, response and SNMPv2 trap PDUs. Thus there is a direct relationship between the actual SNMP Pdu types and how they are represented as classes in JMX.

In the SNMP standard, messages are used to encapsulate PDUs (RFC 1157, RFC 1902). The class SnmpMessage is used to implement this function. It is SnmpMessage objects which are exchanged between peers and thus these represent the lowest level of the communication function. The EncodePdu and DecodePdu methods of the

SnmpPduFactory are called by this class, prior to the information being transmitted on the link.

PACKAGE JAVAX.MANAGEMENT.SNMP.MANAGER

This package contains the interfaces and classes needed to create a SNMP manager. The interfaces and classes are largely concerned with providing the functionality required to allow the manager to control the communication with the agents.

SnmpSession is the main class in the package. Although the concept of a session is not defined by SNMP, in the JMX API, its purpose is to insulate the complexity of the low level communication undertaken by the manager, from the network management application. Essentially the network management application drives the manager by issuing requests for Mib variable information from agents. The manager responds to these requests by first creating a SnmpVarBindList object. This object contains an array of SnmpVar objects. The class SnmpVar extends the SnmpVarBind class discussed earlier and thus fulfils the same purpose, which is to provide details of the object identifier as, defined by the SNMP standard. In this instance, the SnmpVarBindList will contain one SnmpVar object for every Mib variable requested by the network management application.

The SnmpSession Class defines methods for building every type of SNMP Pdu, for instance there is a SnmpGet method which builds a SNMP Get Pdu. Similarly there is a SnmpGetNext method, etc. The manager translates requests from the network management application into the appropriate SNMP Pdu type by calling the appropriate method (SnmpGet, etc) passing the SnmpVarBindList as a parameter. Although the detail is hidden, the method also carries out all the necessary low level communication tasks, for instance encoding and decoding, similar to those explained previously. The

method call also returns a SnmpRequest object, so called because it represents the request from the network management application. Thus every Pdu transmitted has an associated SnmpRequest object. This SnmpRequest object is used to indicate whether synchronous or asynchronous transmission mode should be used to transfer the Pdu and also to store the response from the agent. To operate in synchronous mode, which means the manager waits for a reply before sending the next Pdu, the WaitForCompletion method of the SnmpRequest class is executed. This mode effectively enforces single threading of PDUs. To operate in asynchronous mode, the SnmpRequest object must invoke a method of the class that implements the SnmpHandler interface. The method is called when certain events occur, for instance, a response is received from the agent, or, an error has been detected in the Pdu. They allow the manager identify when certain events have taken place, and to react accordingly.

The manager must create a SnmpSession object before it can begin communication. Within a session, a manager may communicate with one or more agents. The methods provided by the class allow the manager to define the various session parameters, for instance, the agent IP address used in the session. The class also allows the manager to create and customise SNMP PDUs. For instance, it is possible to create a get PDU by using the method SnmpGet. The method performs a single SNMP `get` request on the variable binding list set by its parameter of type SnmpVarBindList.

The SnmpPeer class represents an agent within the manager. It contains agent data (and corresponding methods for manipulating the data) that uniquely identifies it, and allows the manager to establish communication with it. For instance, IP address and port

numbers are two data items defined. It also stores parameters used to control the communication, for instance the number of retries and timeout values.

A trap function, which is similar in concept to interrupts used by operating systems, is defined in the SNMP standard. A SNMP agent issues a trap when a significant and unexpected event occurs. For instance, the agent may report the failure of a communication link using a trap. The SnmpEventReportListener interface together with the SnmpEventreportDispatcher class is the means by which traps are implemented in the JMX API. The interface defines the processSnmpTrapV1 and processSnmpTrapV2 methods. The manager invokes these when a trap is received. The class implementing the interface must register it with an instance of the class SnmpEventReportDispatcher. An instance of this class initialises a thread that waits for incoming traps from an agent. When a trap is received, the appropriate trap method is called.

The agent and manager exist to control the exchange of SNMP messages. The JMX API packages javax.snmp.manager and javax.management.snmp.manager provide the interface and classes required to implement SNMP agents and managers respectively and the preceding sections gave an explanation of the interfaces and classes contained in the packages. The aim was not to describe every interface and class in detail, but rather to focus on those aspects which are central to understanding how communication between the agent and manager takes place.

JDMK PACKAGES

PACKAGE COM.SUN.JDMK.SNMP

This package contains the classes which implement the interfaces SnmpOidTable and SnmpPduFactory. These interfaces are part of the JMX API standard library package

javax.management.snmp and an explanation of them has already been given. Note that the classes are used to develop a SNMP manager. The package also contains the interface SnmpOidDatabase and the class which implements this, SnmpOidDatabaseSupport. This class provides the facility for supporting multiple MIBs.

PACKAGE COM.SUN.JDMK.SNMP.AGENT

The interfaces and classes needed to develop a SNMP agent are contained in the package com.sun.jdmk.snmp.agent. Altogether it contains three interfaces and eight classes.

The interface SnmpMibAgentMBean represents the template used to develop a SNMP agent and is implemented by the class SnmpMibAgent. It contains methods corresponding to the SNMP operations, for instance the GetPdu method corresponds to the SNMP get operation. These methods provide the logic for processing the PDUs, including retrieval of the relevant information from the agent's Mib. They are invoked when the corresponding Pdu is received from the Manager.

The interface also defines methods which enable the agent to manipulate protocol adaptor objects. It particular the SetSnmpAdaptor method provides the means for an agent to store a copy of the protocol adaptor object reference. This provides a direct link from the agent to the protocol adaptor object and thus allows the agent to invoke methods on the adaptor object.

SnmpMibHandler is the interface which provides the template for the protocol adaptor. In JDMK the class SnmpAdaptorServer implements this interface. The SnmpMibHandler defines the AddMib method which, as the name suggests, adds

details of a Mib to the protocol adaptor. Effectively the Mib object reference is stored in the protocol adapter object which invokes the AddMib method. This allows the adaptor to invoke methods on the agent. It is important to note that the JDMK API documentation explicitly states that this method 'is called automatically by the SnmpMibAgent.setSnmpAdaptor method and should not be called directly. How this automatic invocation is achieved is explained shortly when the SetSnmpAdaptor method is explained.

As noted above, the SetSnmpAdaptor method is responsible for invoking the AddMib method in the SnmpAdaptor Server class.

The remaining interfaces and classes in the package are concerned with the building and maintenance of the agent Mib. It is important to appreciate the manager uses the Mib only as a type of dictionary, to validate Mib variables. Thus the manager Mib stores the Mib variable identifiers, their related syntax rules, and the relationship between the variables, which is captured by the hierarchical structure. While the agent also uses the Mib as a dictionary, it must also store the current values of the Mib variables. Thus the agent Mib requires data structures, including tables, for storing the variable values. The generation of these data structures and their subsequent manipulation (as Mib variable values are added and retrieved) inevitably means the agent Mib is more complex.

There are a number of classes provided for building the agent Mib. The class SnmpMib, which extends the class SnmpMibAgent, is used to create the hierarchical (tree) Mib structure in the agent. The nodes at the bottom level of the structure are objects of class SnmpMibNode. This class has two subclasses, namely SnmpMibOid and SnmpMibTable. Essentially Mib variables are either created as SnmpMibOid objects (if they have simple scalar values) or SnmpMibTable objects (if there are multiple and / or

repeating values). The SnmpMib class contains a variable root (which is a SnmpMinOid object); this is the reference to the top level of the Mib structure. The lower levels are generated in the usual recursive manner using the RegisterNode method of the class SnmpMibOid. This method takes an object of class SnmpMibNode as a parameter and adds it to the tree. Clearly the class SnmpMibNode must have at least one variable of class SnmpMibNode which will point, for instance, to the next node in the structure; however precise details of how the tree structure is defined are hidden in the JDMK API.

The class SnmpMibTable implements the SnmpMibTableMBean interface. This class provides the methods for maintaining the Mib variable tables, specifically adding, removing and retrieving entries (rows) in Tables. Each Table entry has two parts, the Index, and a reference to the Object containing the data associated with this Index. The Index is comprised of one of more Mib variables and the Object containing the associated data also comprises one or more Mib Variables. For instance, details of the IpAddrTable table entry (as defined by the SNMP standard) would be stored as follows:

- IpAdEntIfIndex. Internal index
- EntryRef. Reference to an Object containing the fields
- IpAdEntAddr.
- IpAdEntNetMask.
- IpAdEntBcastAddr.

Although the precise detail of how table entries are grouped to form tables is hidden, an array of entries is probably the most obvious solution. The AddEntry method illustrates how table entries are stored. The method takes two parameters, one of type SnmpIndex

and the other of type Object. The SnmpIndex parameter is stored in the next available position in the array together with and the reference to the Object parameter. The SnmpIndex class represents an Index as an array of long numbers.

Note that the manager can also remotely create and remove entries in the agent Mib by using the SnmpMibTableRemCreate class which extends the SnmpMibTable class.

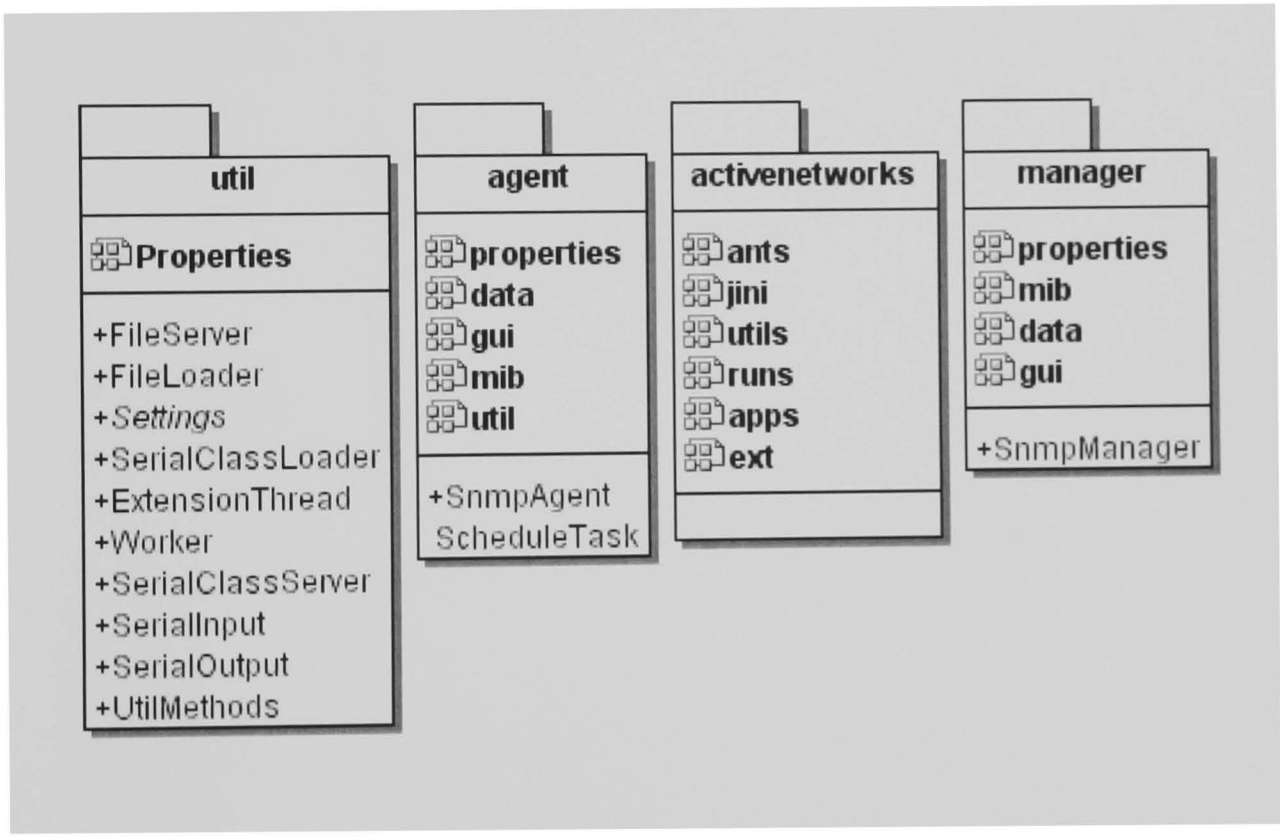
PACKAGE COM.SUN.JDMK.SNMP.IPACL

This is the final package defined by JDMK and it is concerned exclusively with access control aspects of security. Details of the community password and the access rights of managers are stored in a file. This file is either store in a predefined location on the system, or, the location is defined at runtime using the jdmk.acl.file property. The class JdmkAcl, which implements the interface IPAcl, contains the methods used by the agent to check the access rights of the manager against those stored in the file, before granting or denying access. The access control file is associated with the agent when the agent is instantiated, using the SnmpAdaptorServer class constructor.

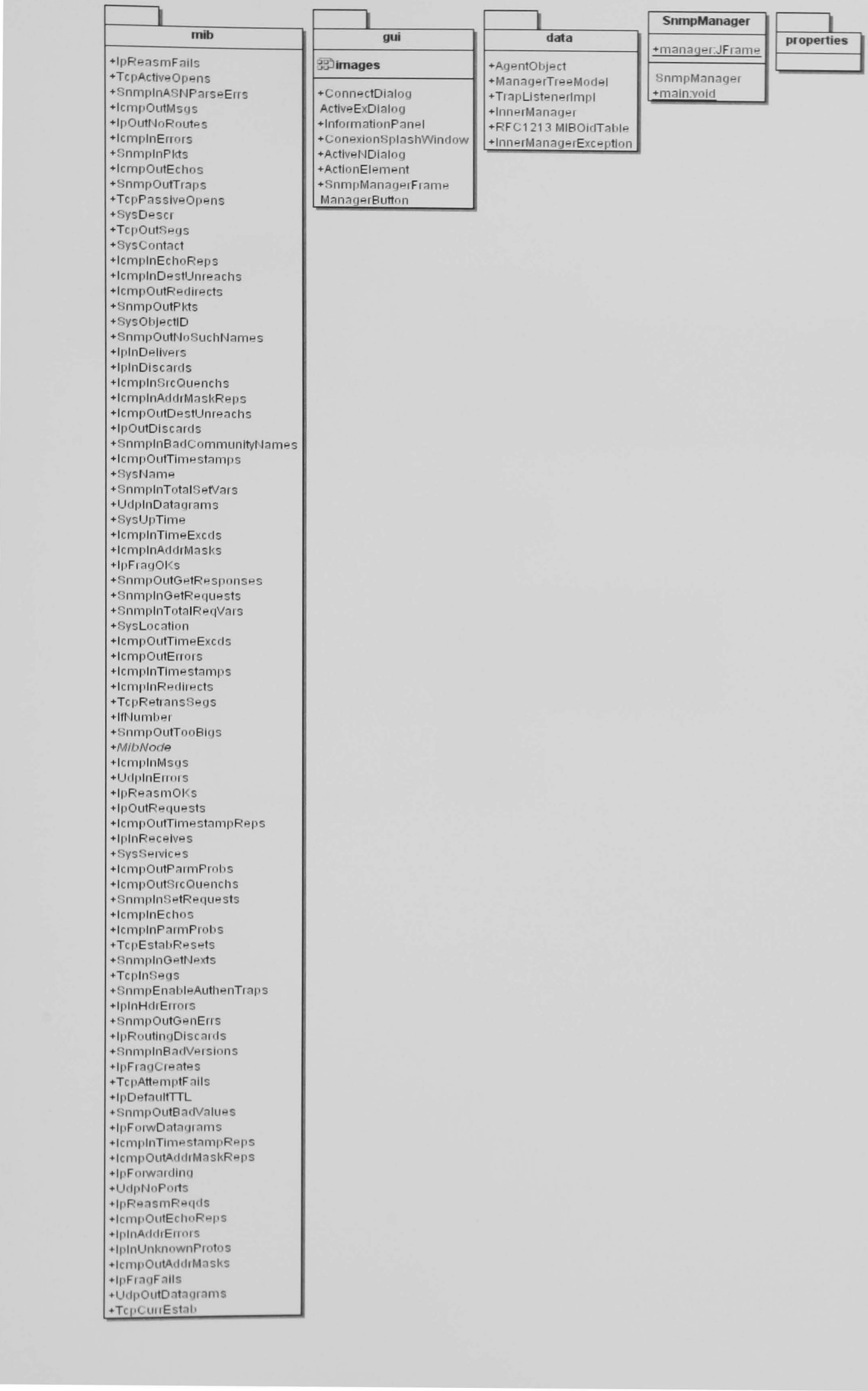
Appendix C

SNMP ACTIVE NETWORKS TOOLKIT DESIGN CODE

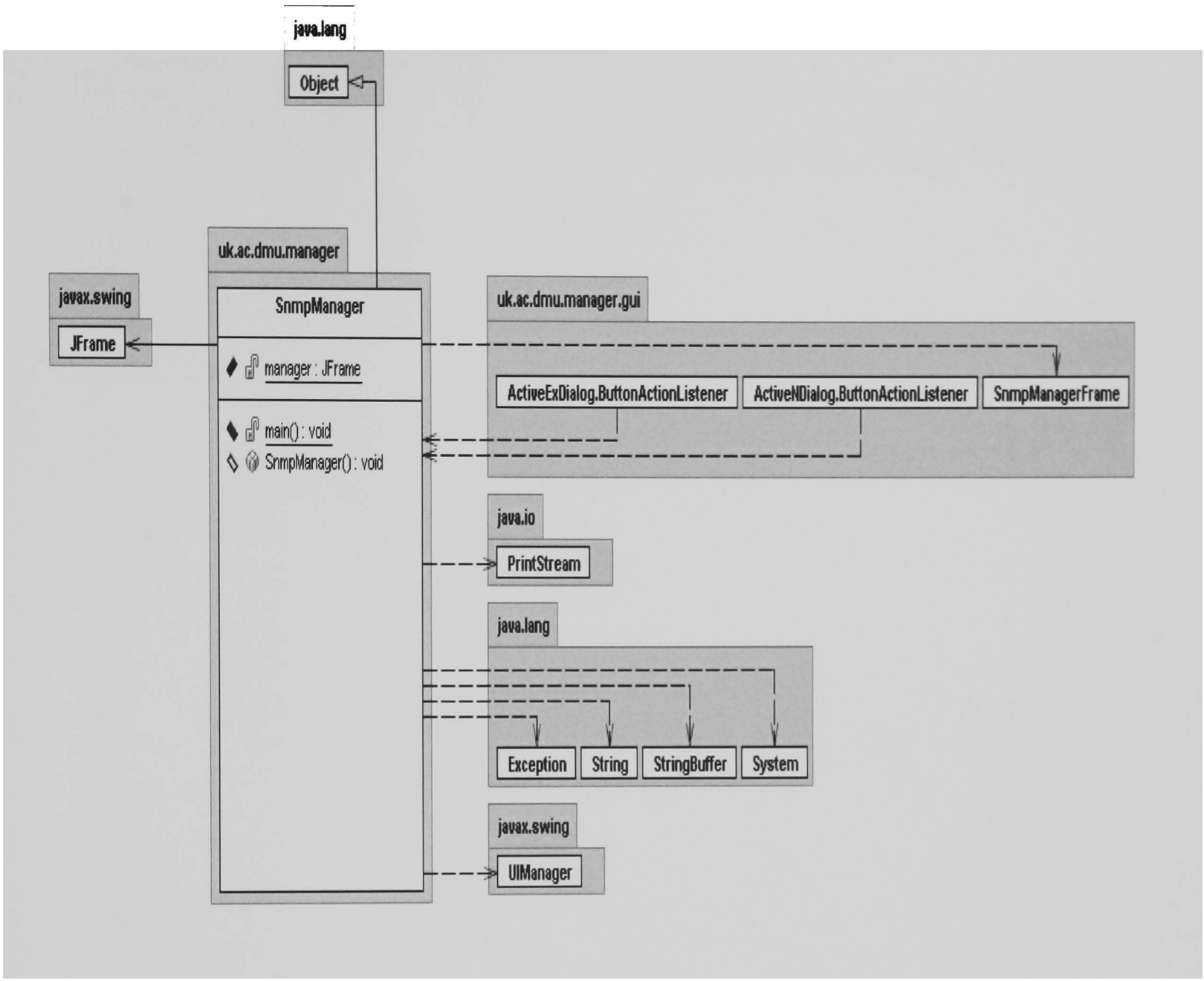
This appendix will provide the complete SNMP Active Networks Toolkit UML diagrams. The ANTS classes (with the changes produced by the Toolkit) will be included as well. The appendix structure will follow the Java packages Toolkit structure, starting with the "uk.ac.dmu.manager" Java package, next will be the "uk.ac.dmu.agent" Java package, following with the "uk.ac.dmu.util" Java package, to finish with the "uk.ac.dmu.activenetworks" Java package. Next picture will provide an overview of the packages integrated in the Toolkit.



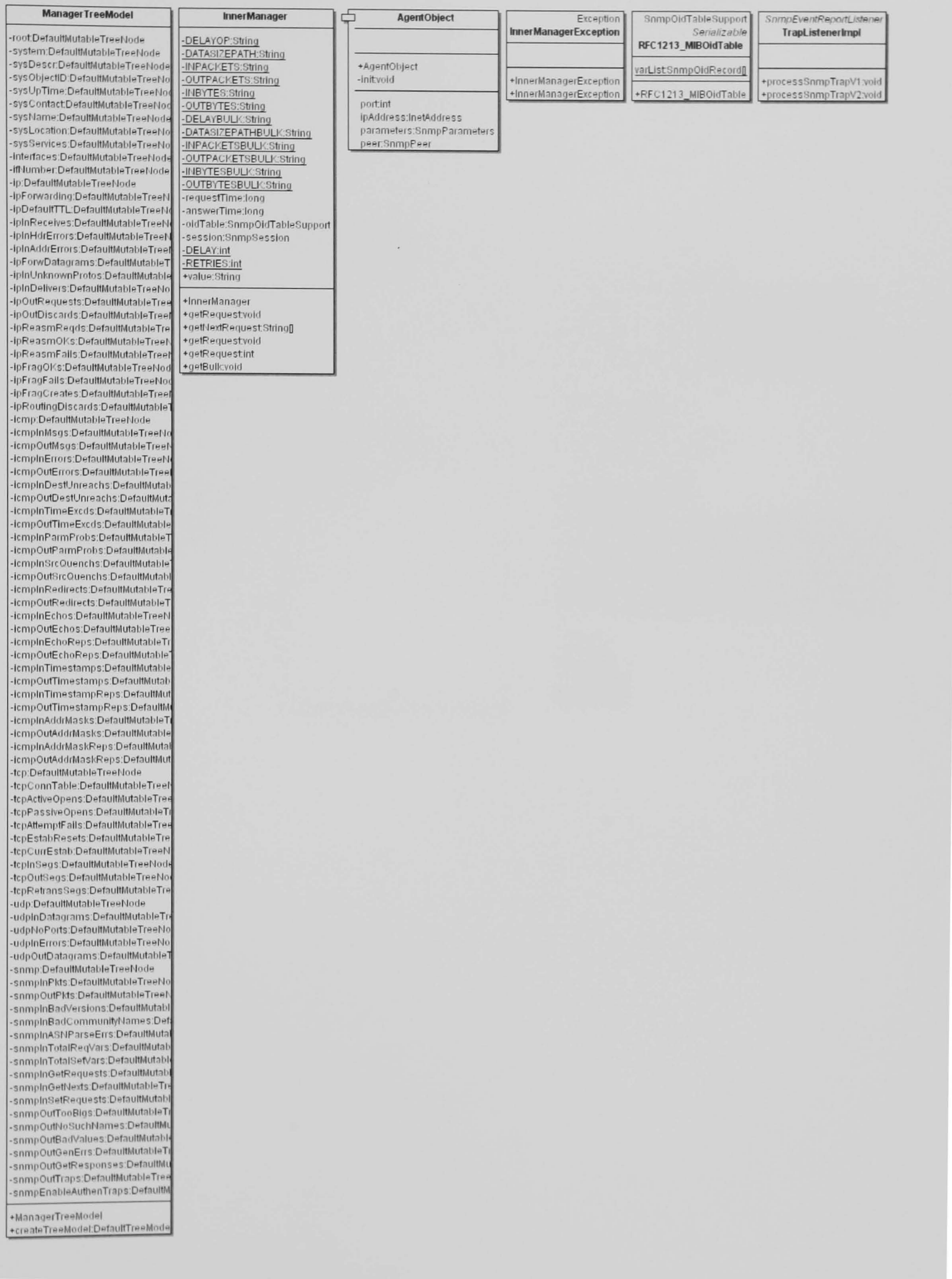
C.1 PACKAGE UK.AC.DMU.MANAGER



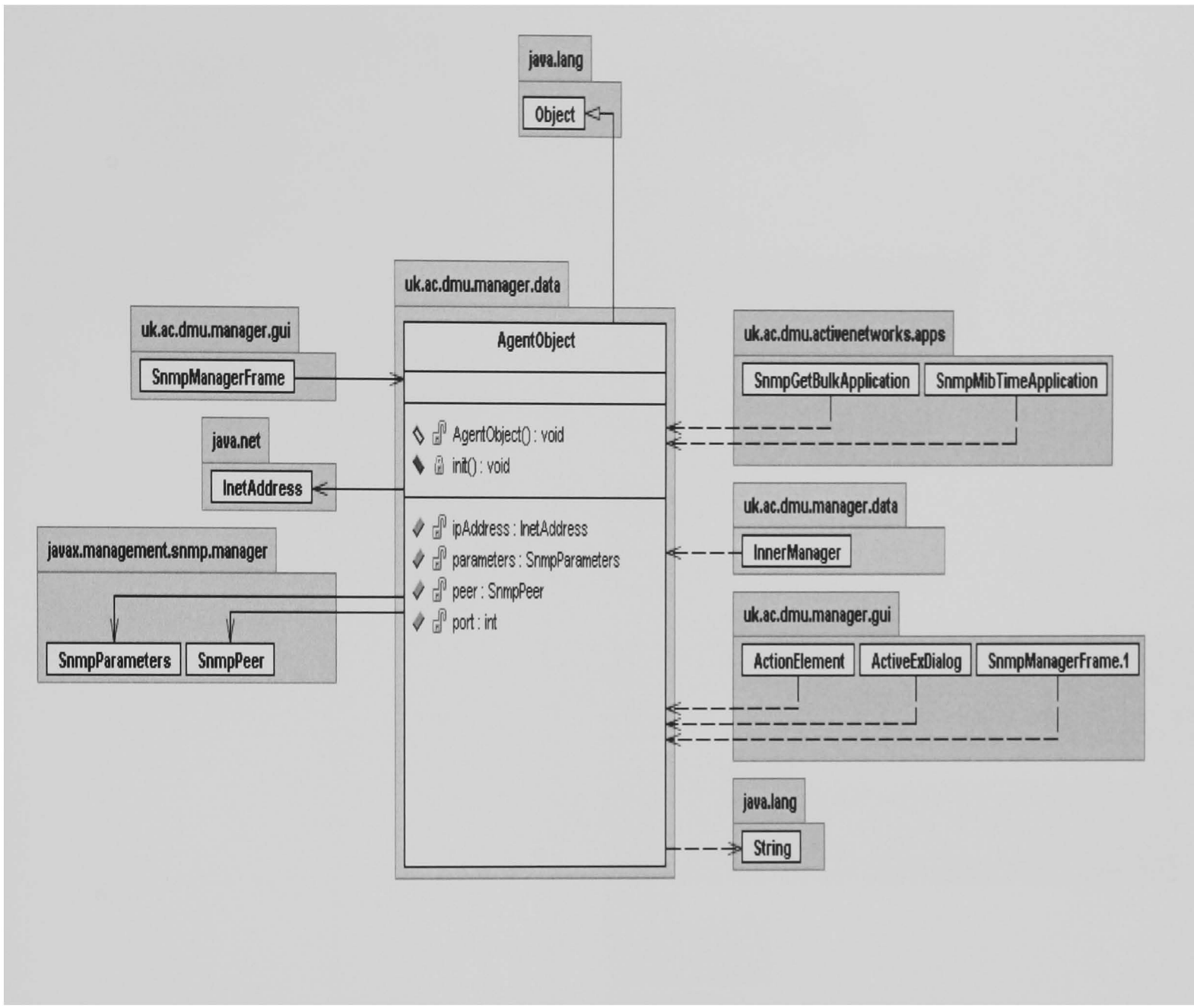
Class: SnmpManager



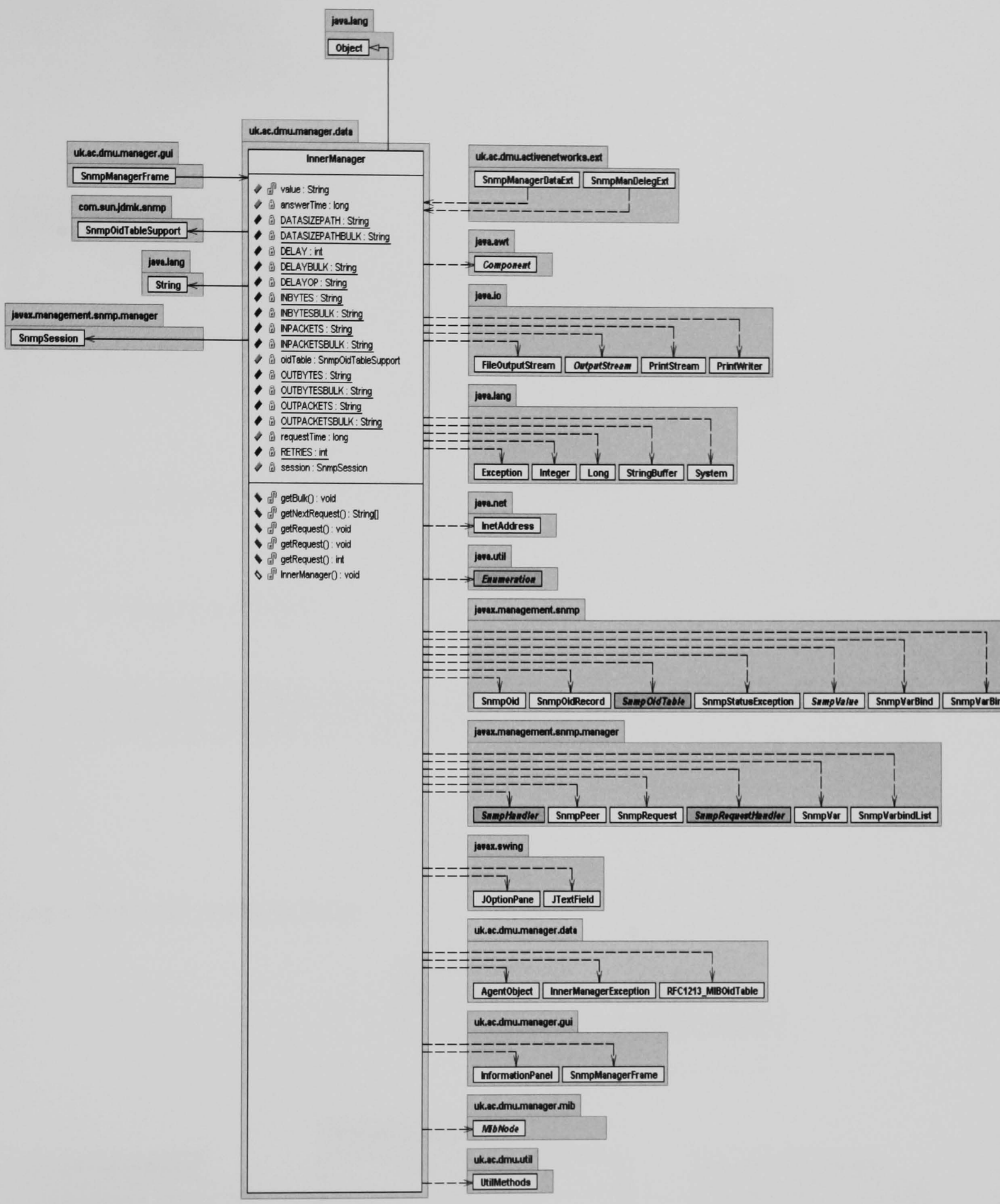
Package: uk.ac.dmu.manager.data



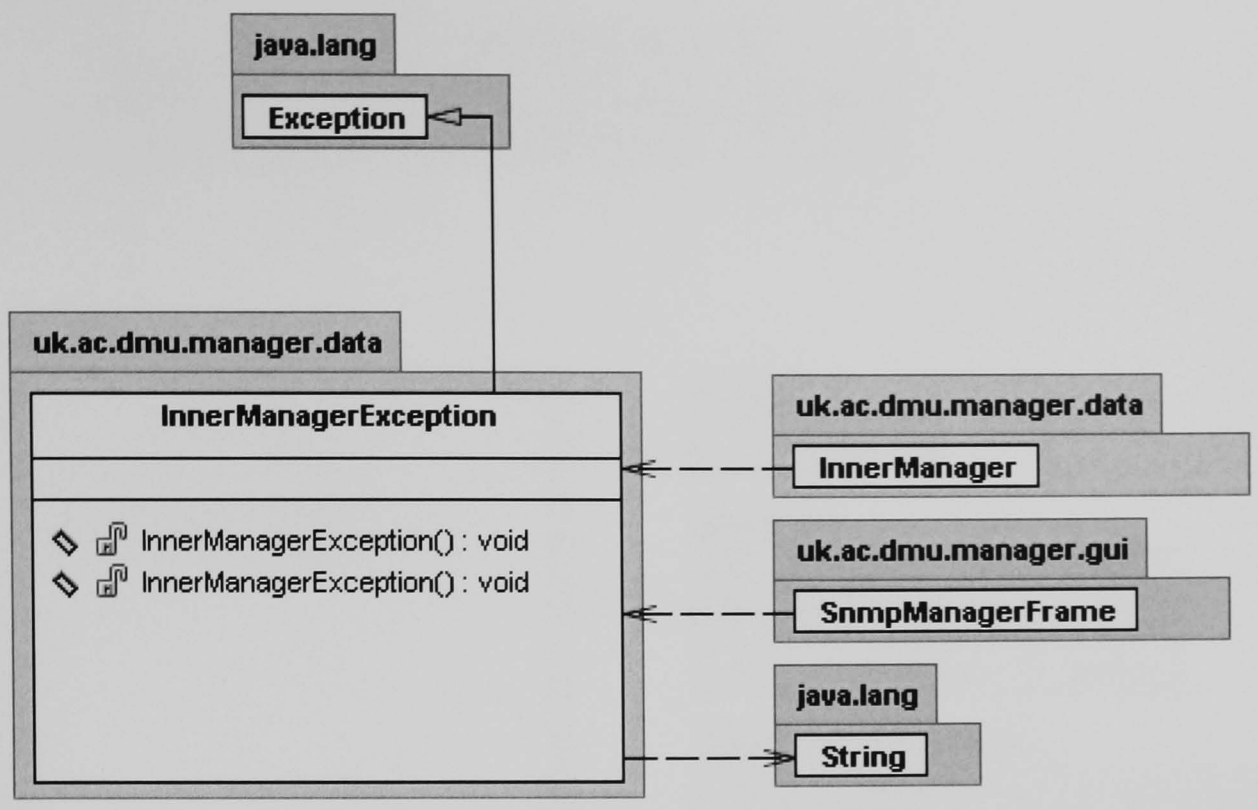
Class: AgentObject



Class: InnerManager

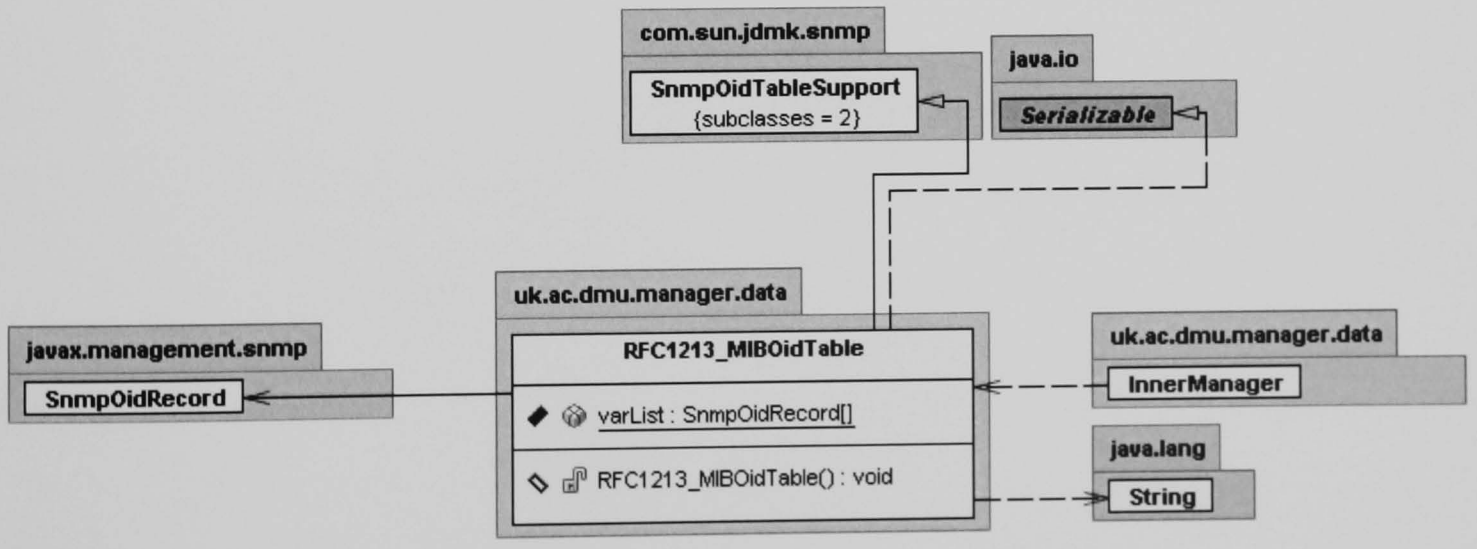


Class: InnerManagerException

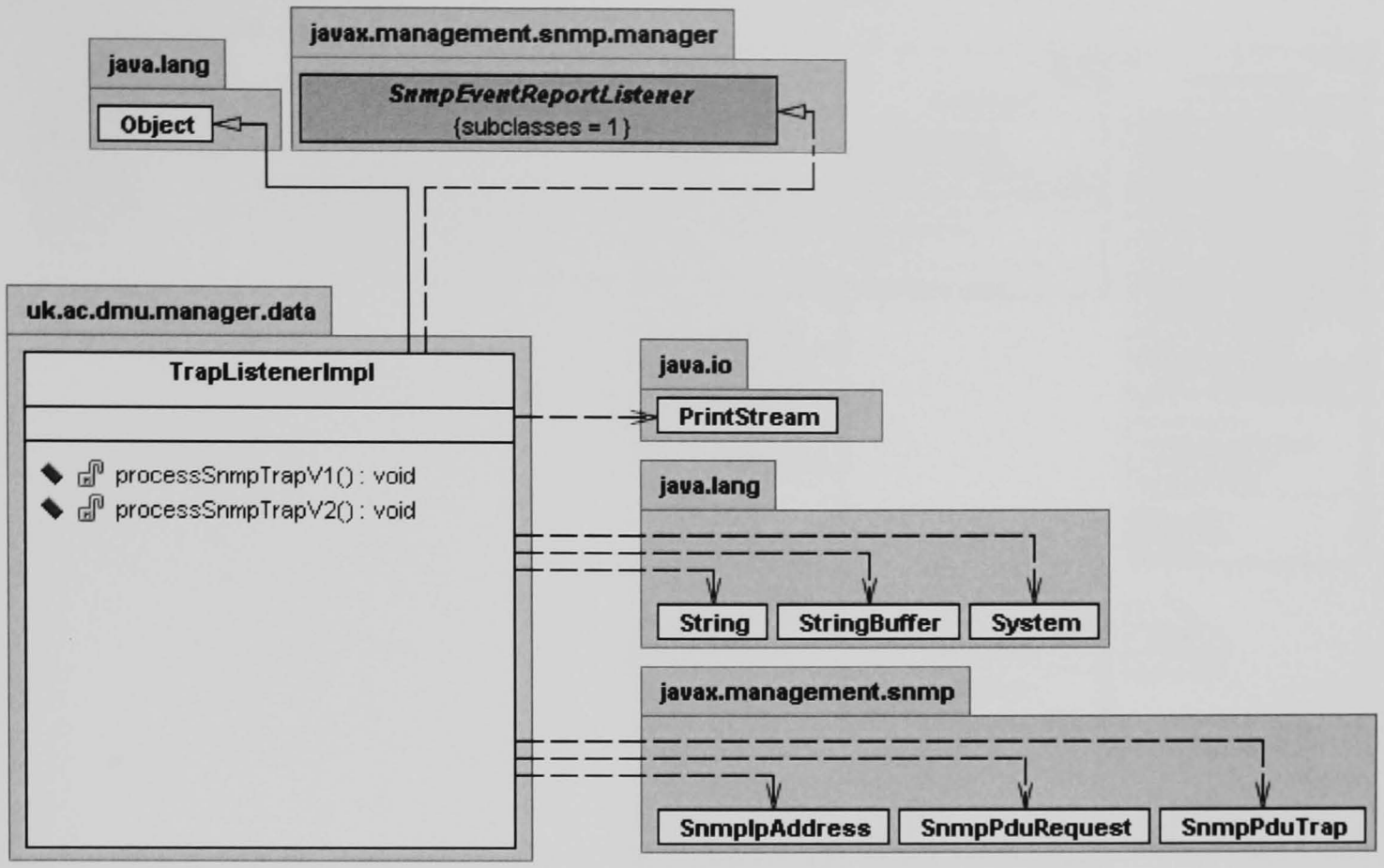


Class: ManagerTreeModel

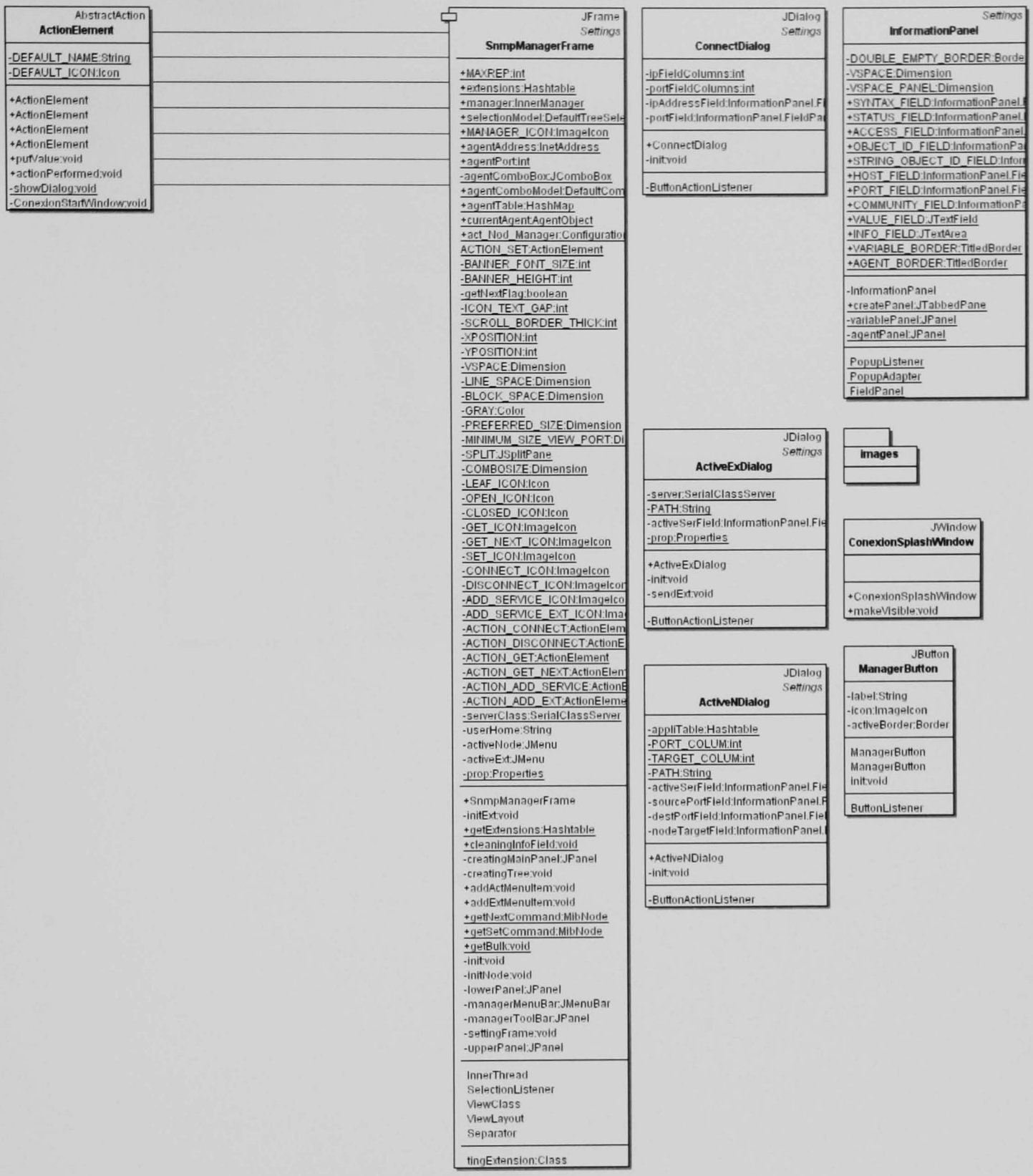
Class: RFC1213_MIBOidTable



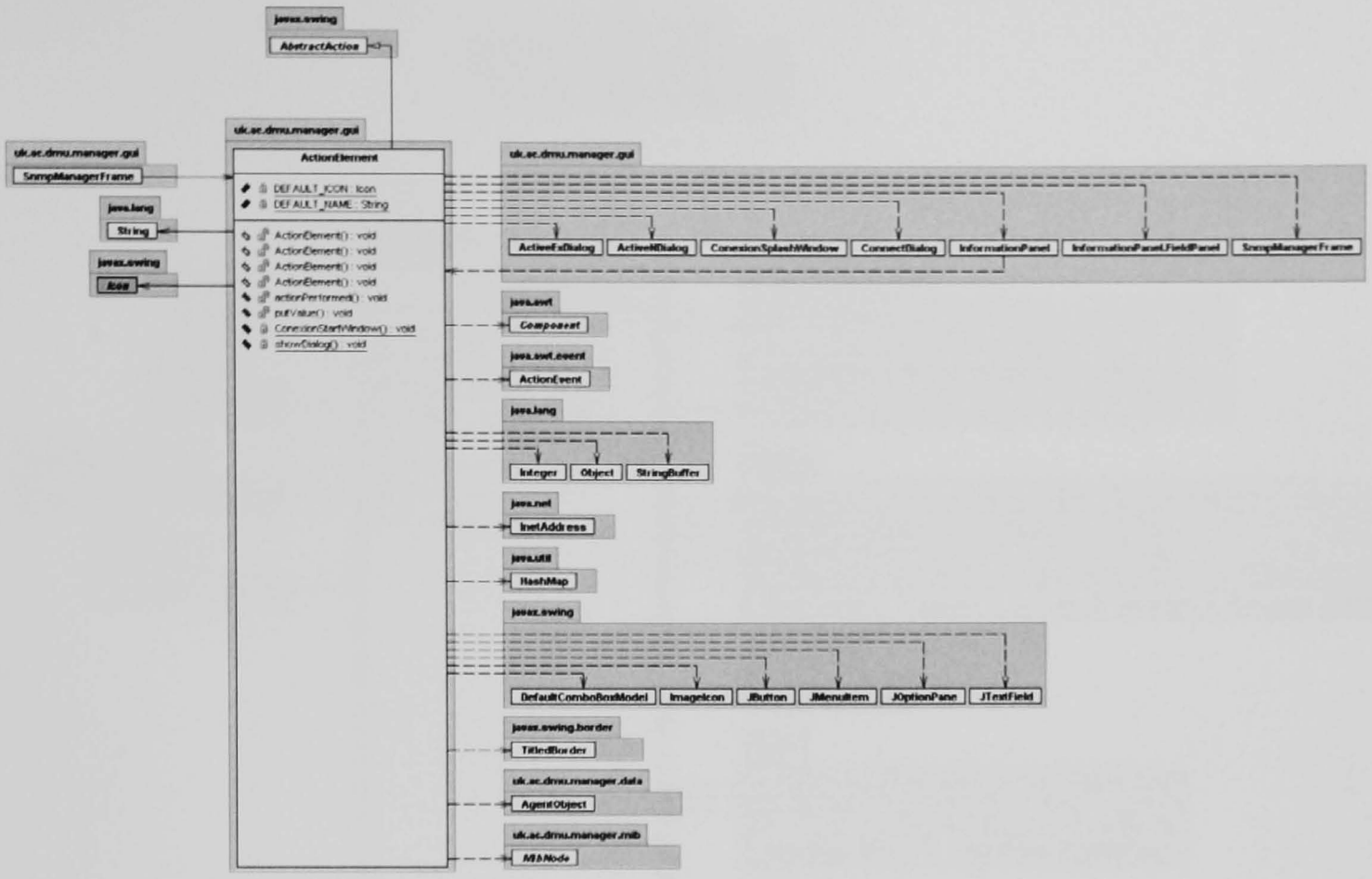
Class: TrapListenerImpl



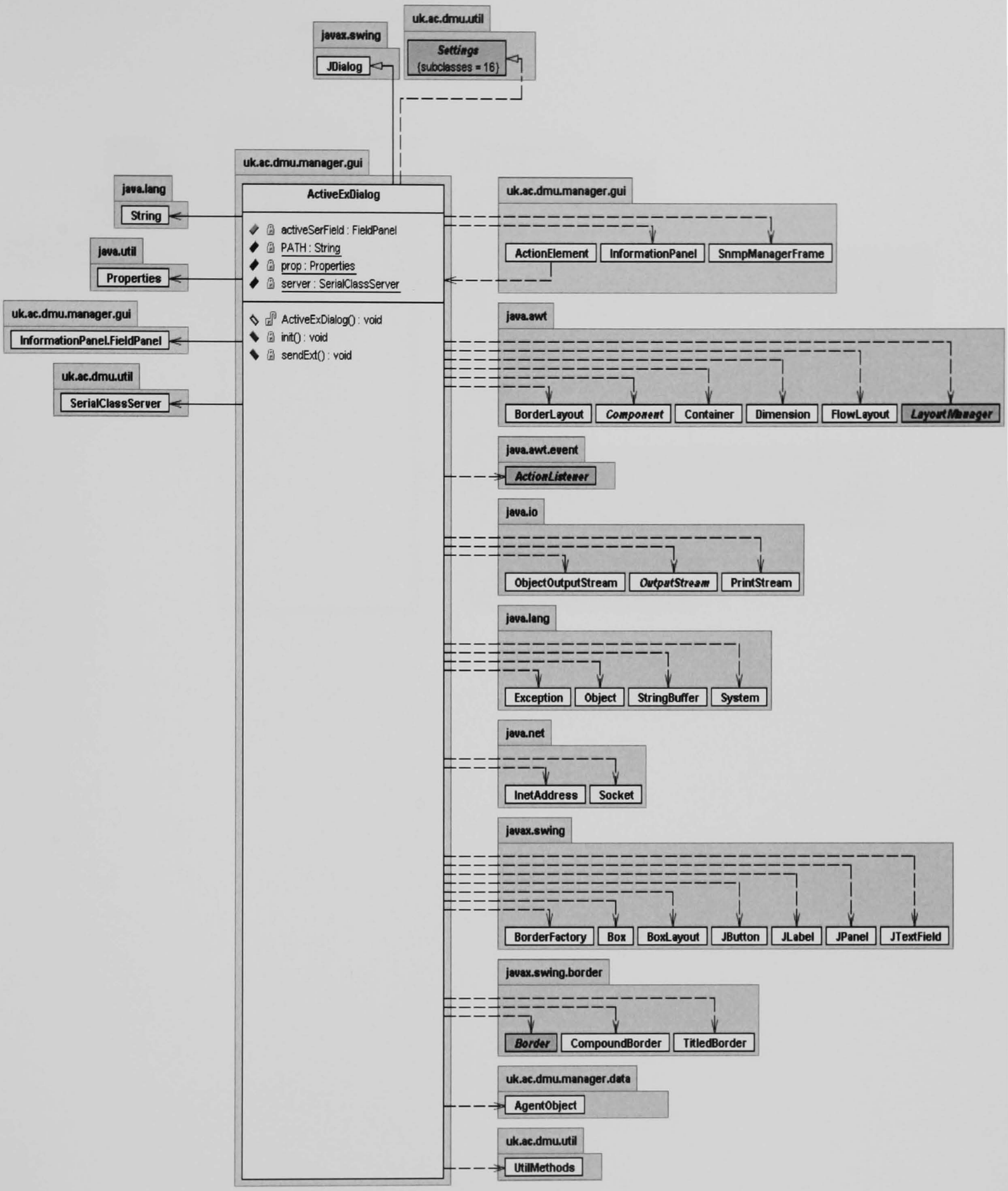
Package: uk.ac.dmu.manager.gui



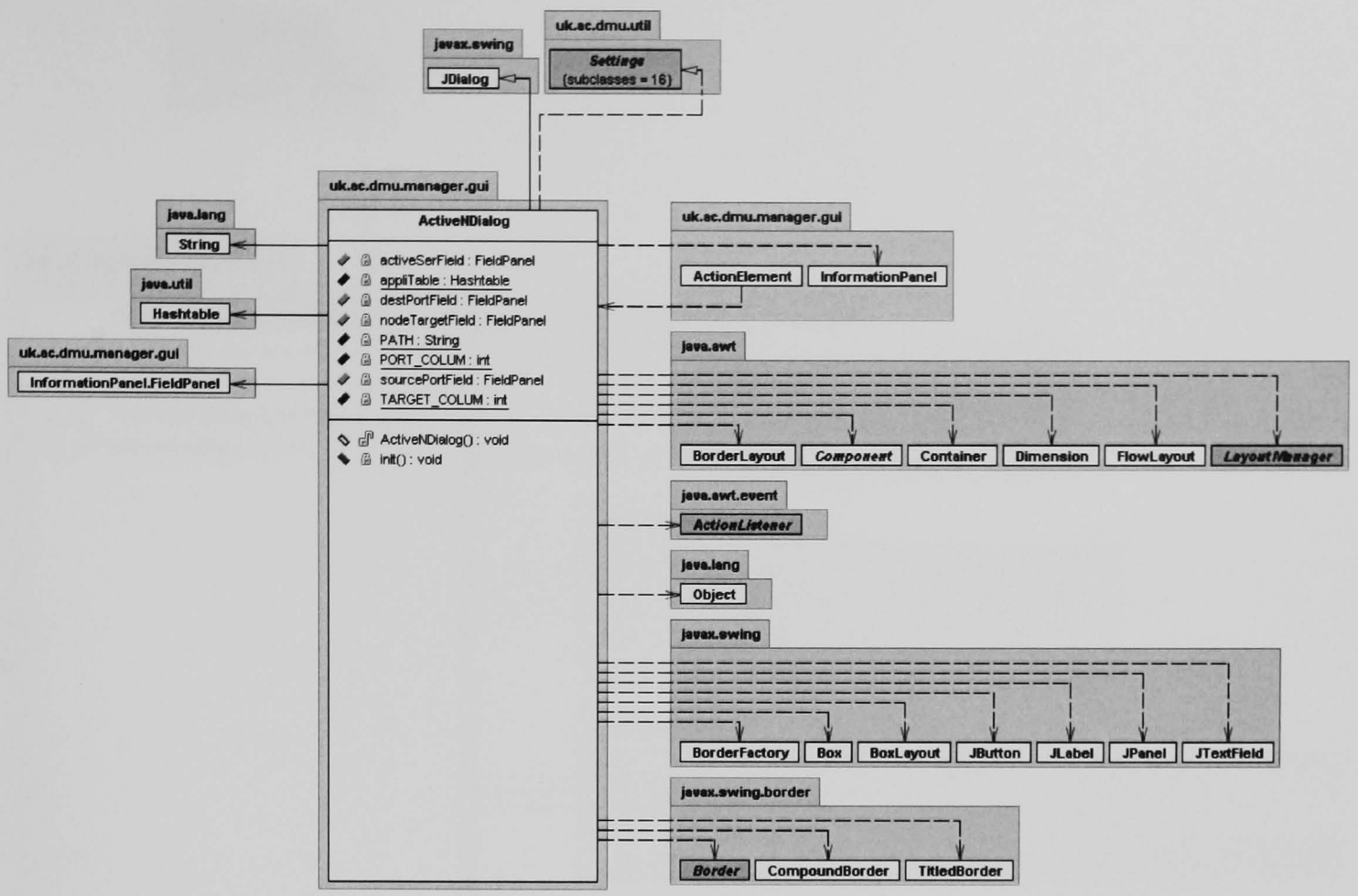
Class: ActionElement



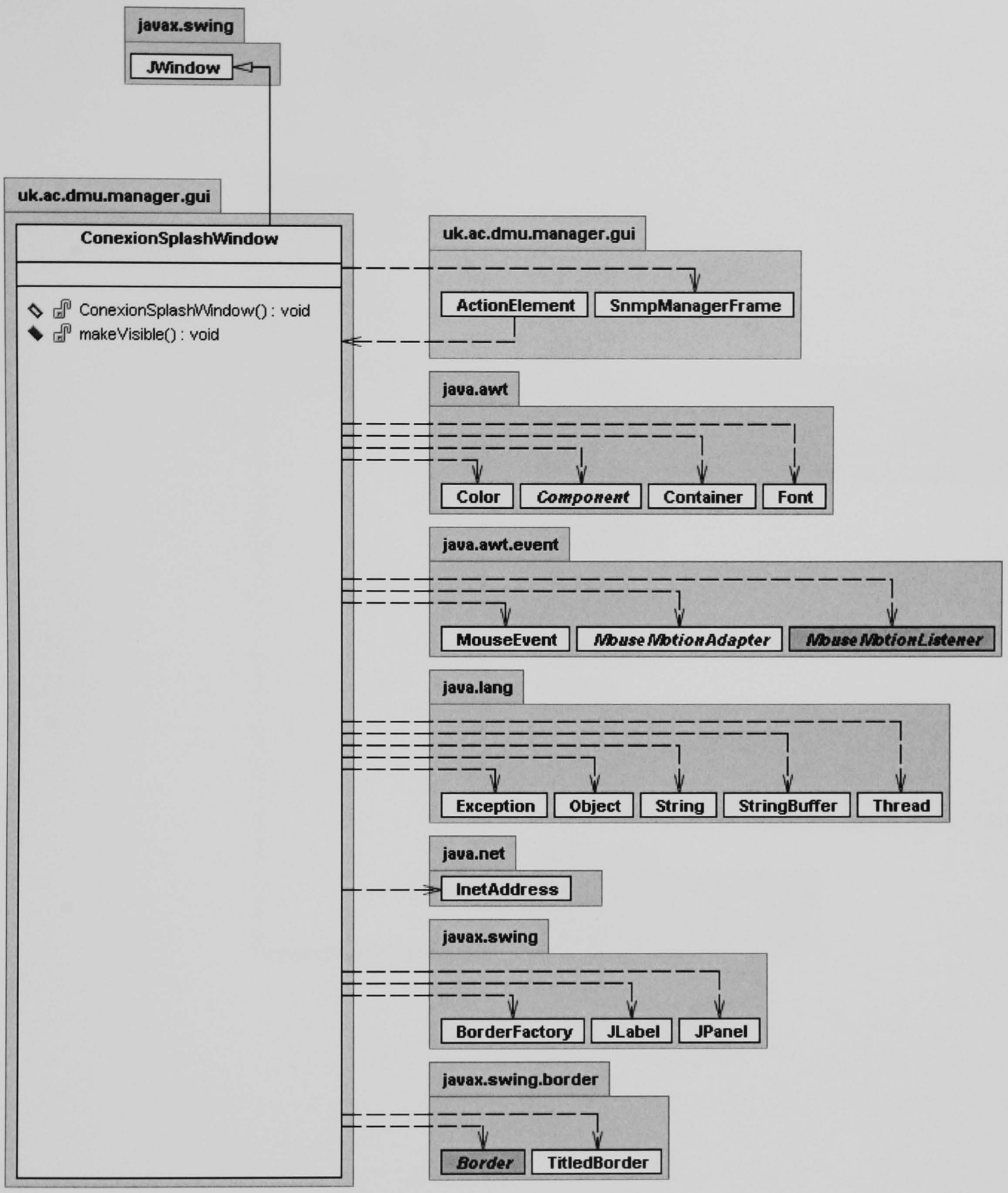
Class: ActionExDialog



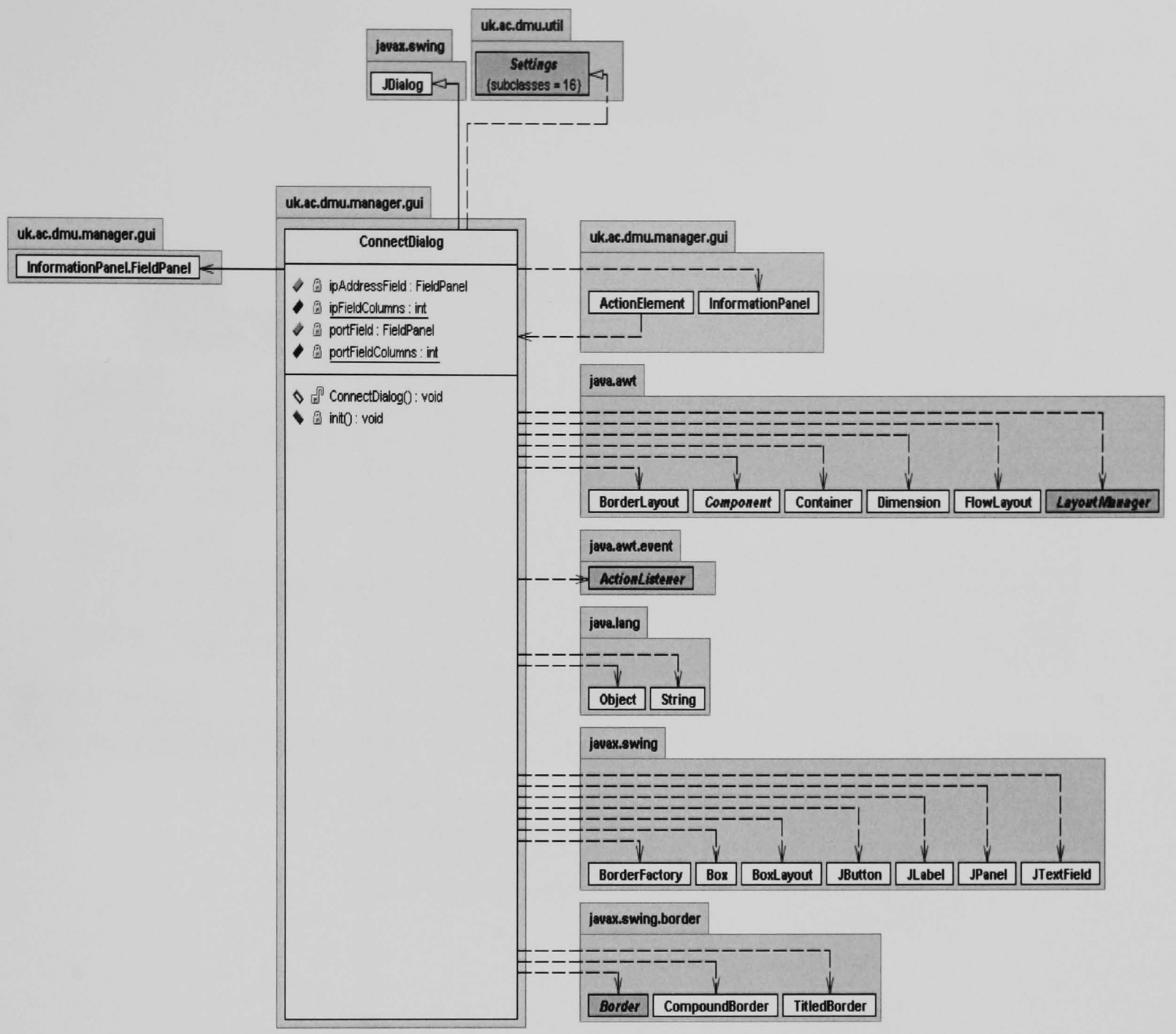
Class: ActionNDialog



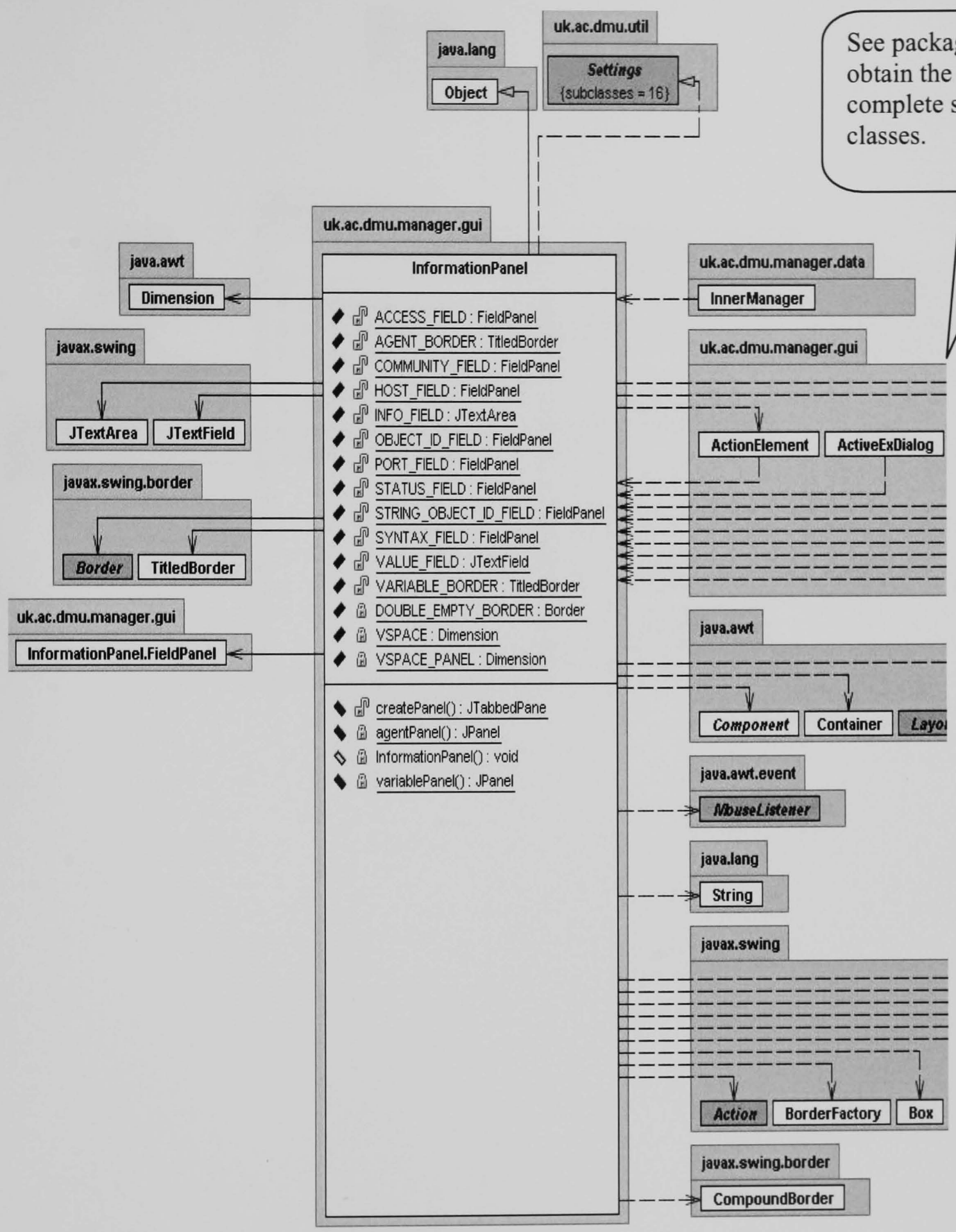
Class: ConexionSplashWindow



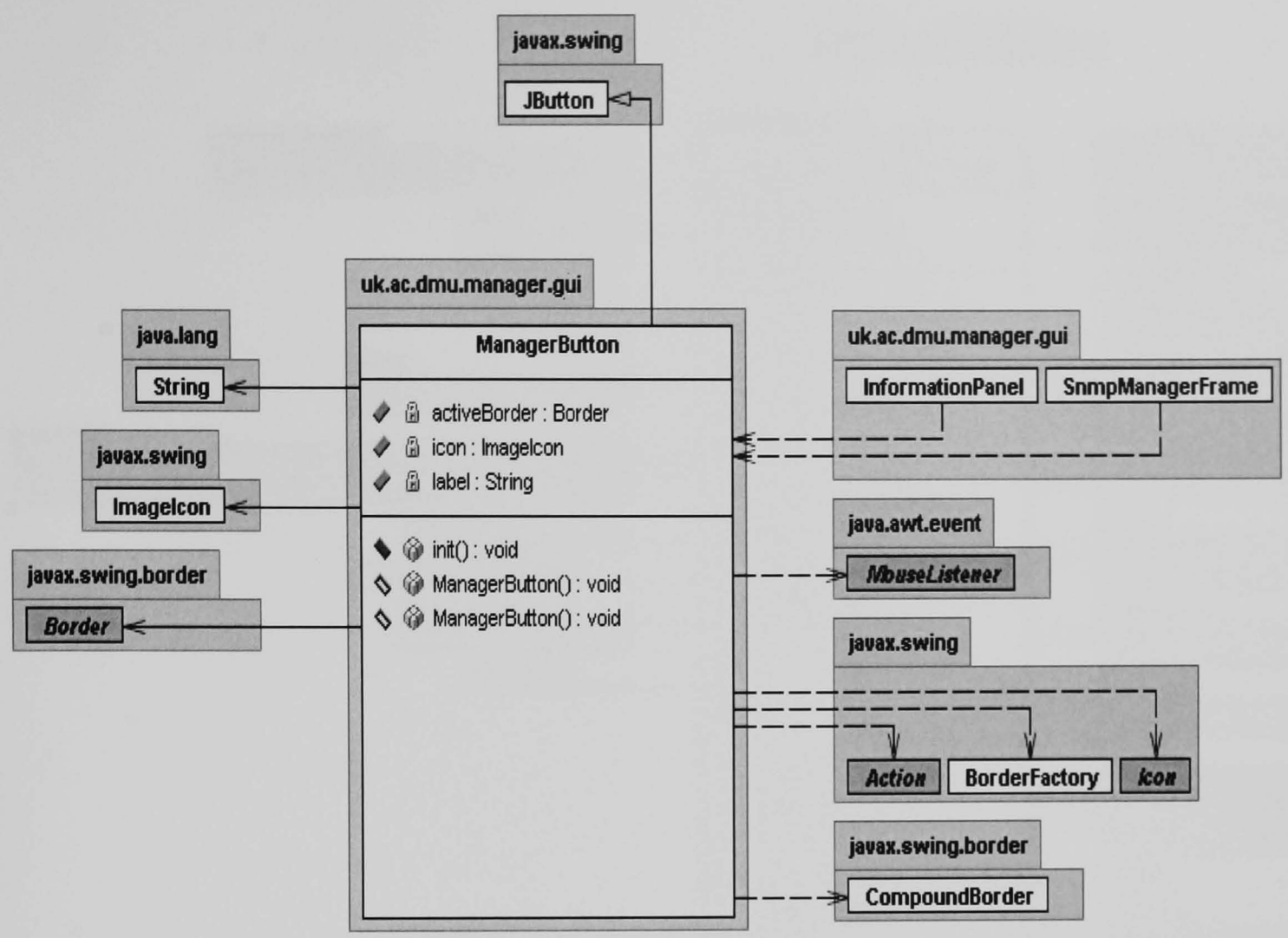
Class: ConnectDialog



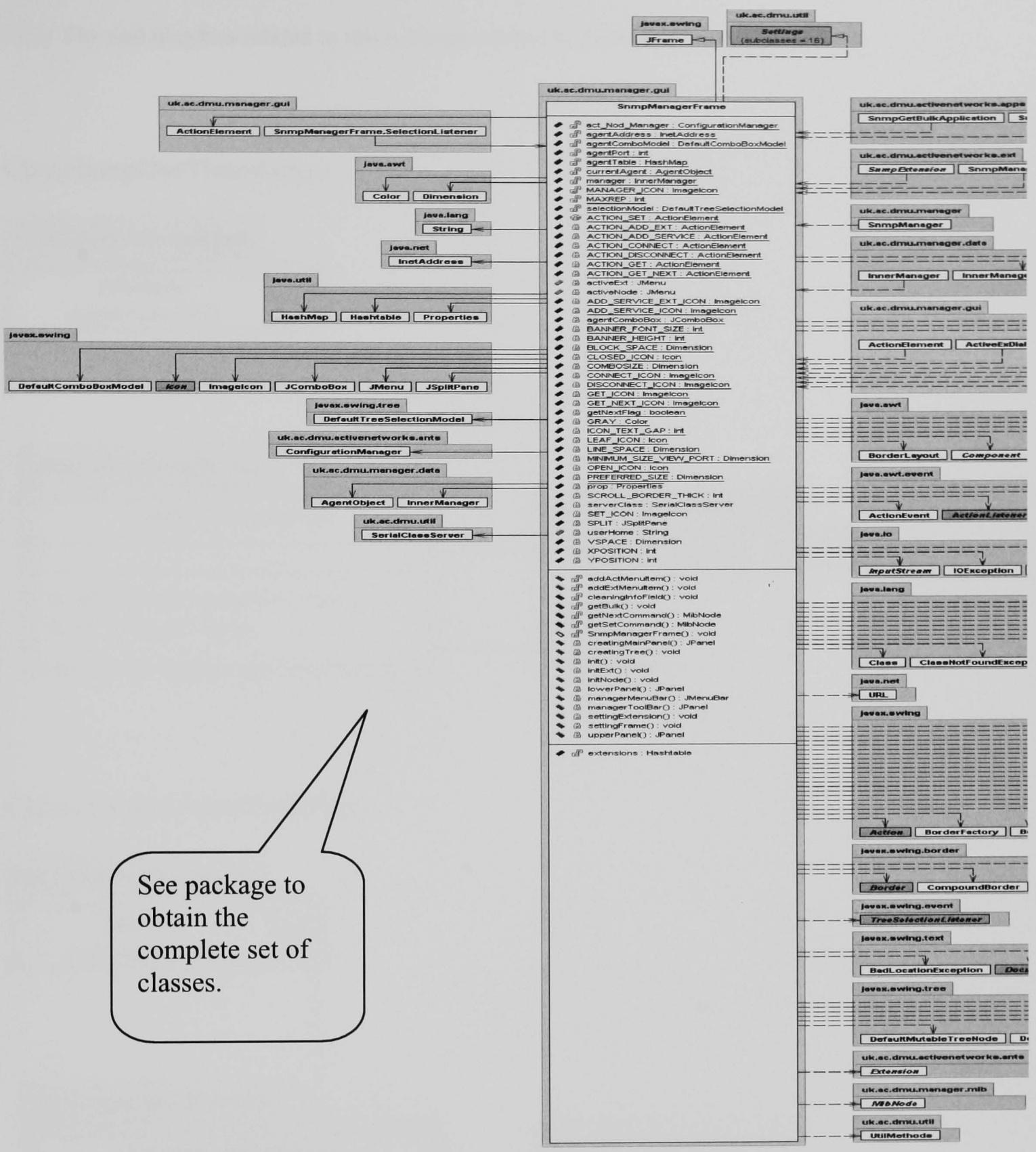
Class: InformationPanel



Class: ManagerButton



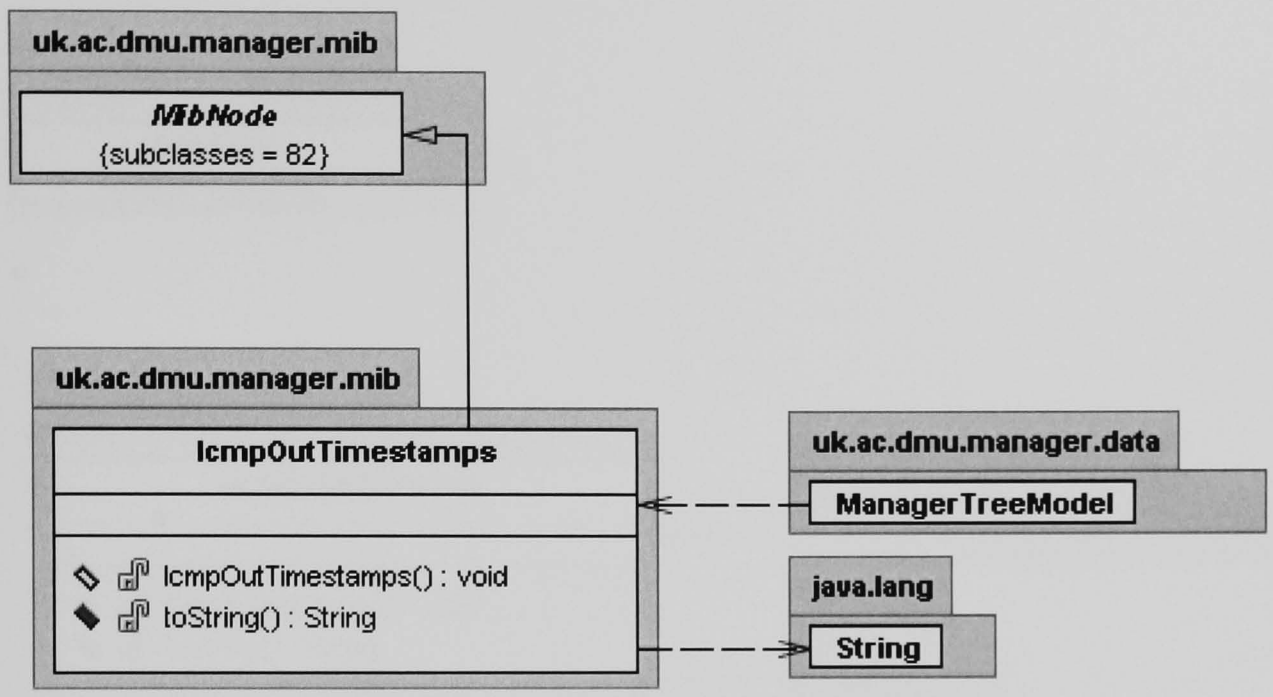
Class: SnmpManagerFrame



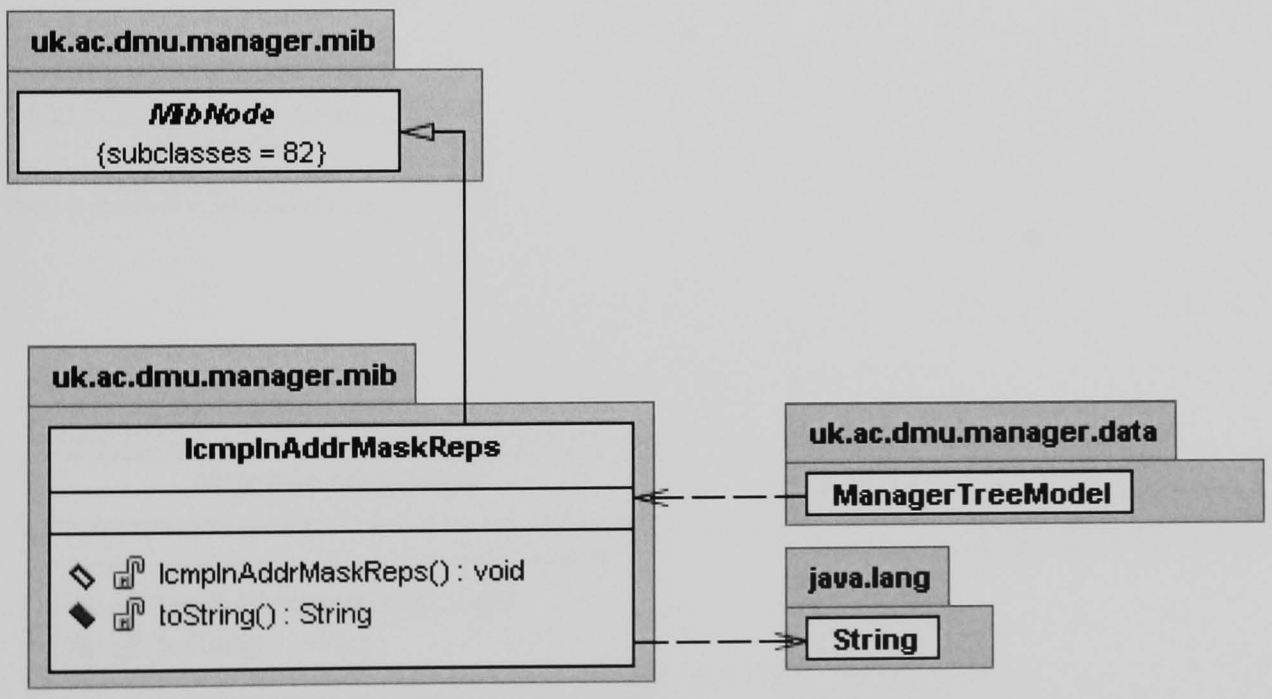
Package: uk.ac.dmu.manager.mib

Note: The uml diagram related to this package can not be formatted well to be printed.

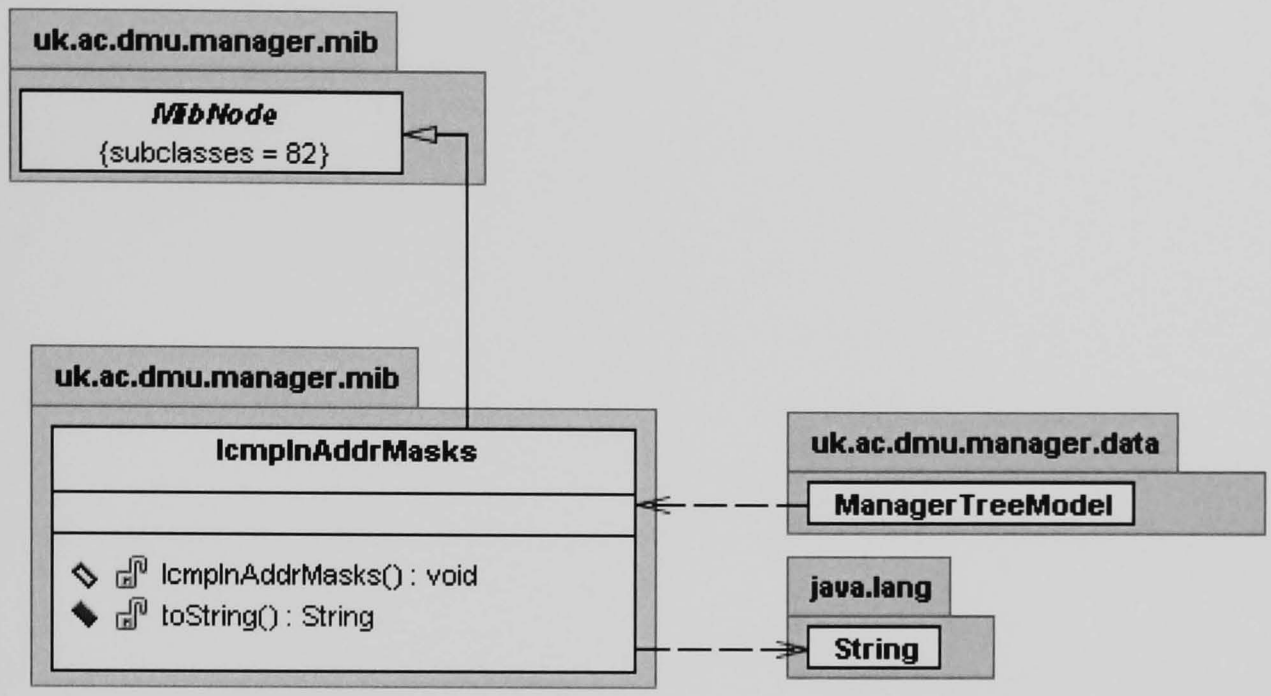
Class:IcmpOutTimestamps



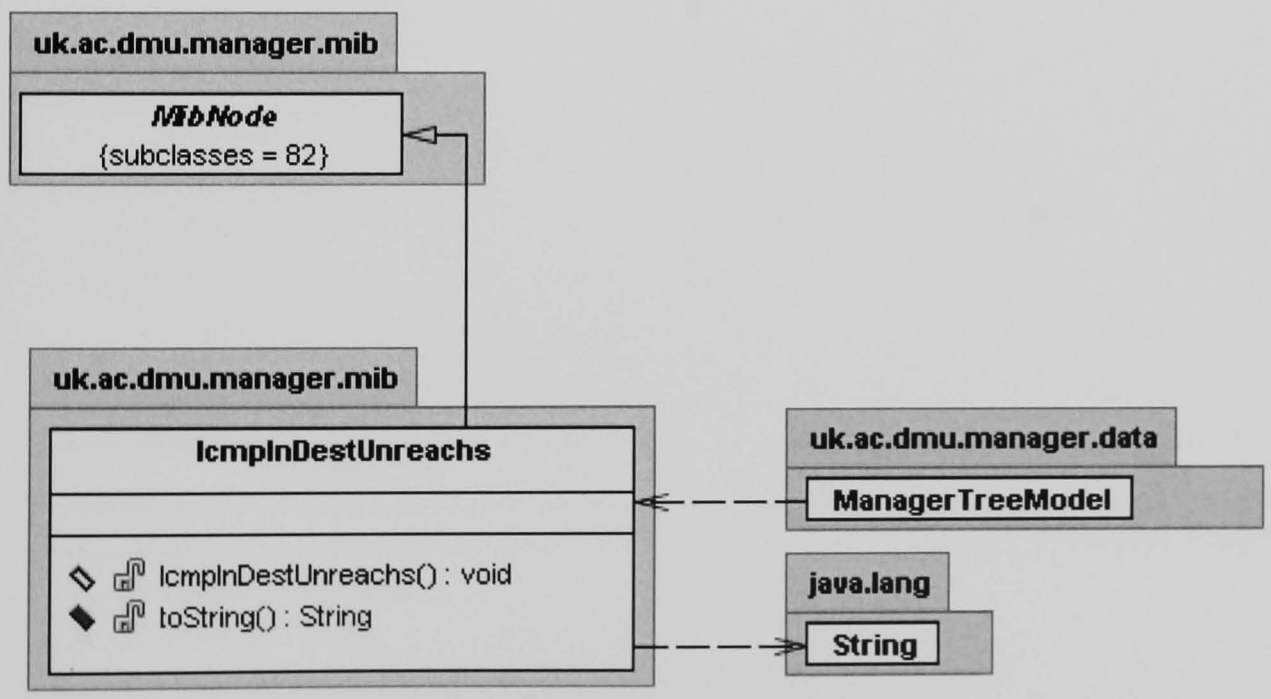
Class: IcmpInAddrMaskReps



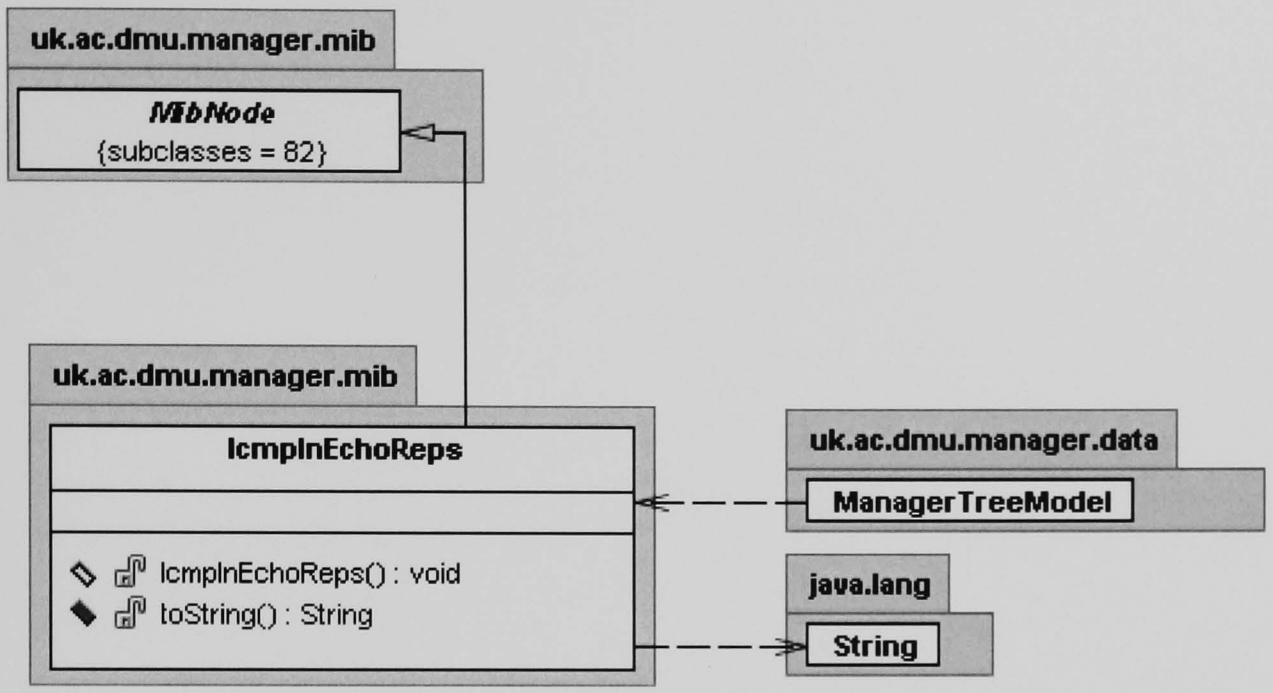
Class: IcmpInAddrMasks



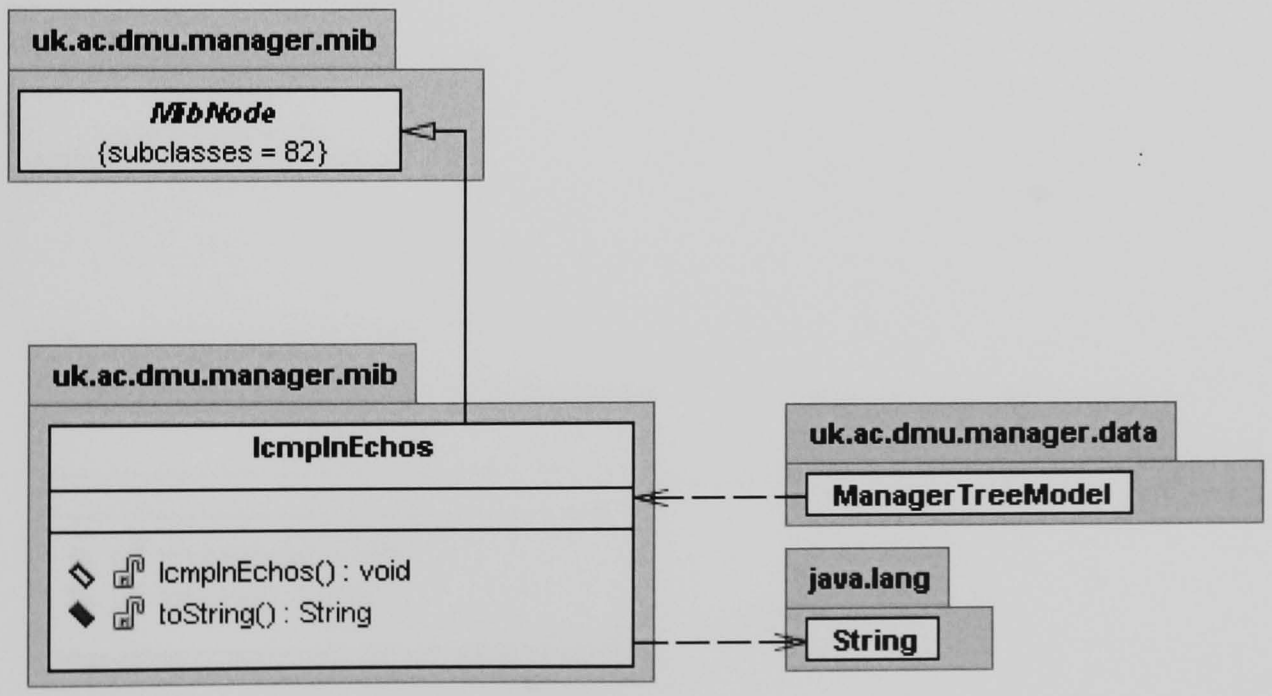
Class: IcmpInDestUnreachs



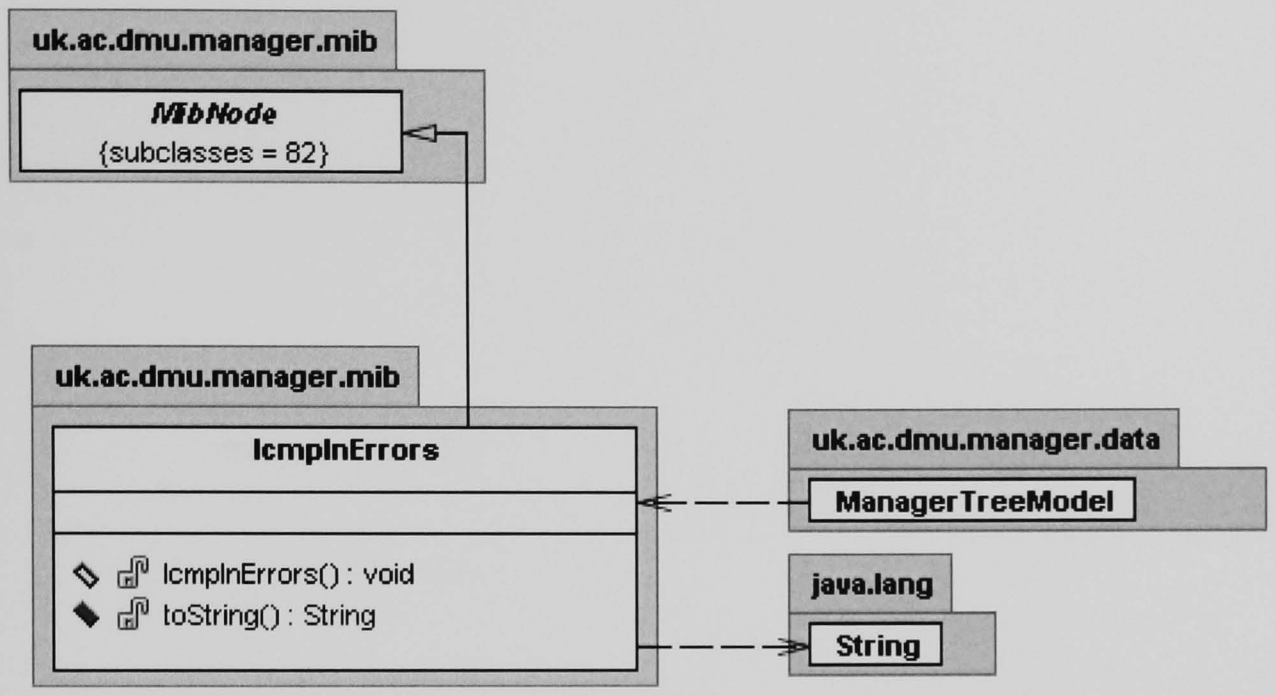
Class: IcmpInEchoReps



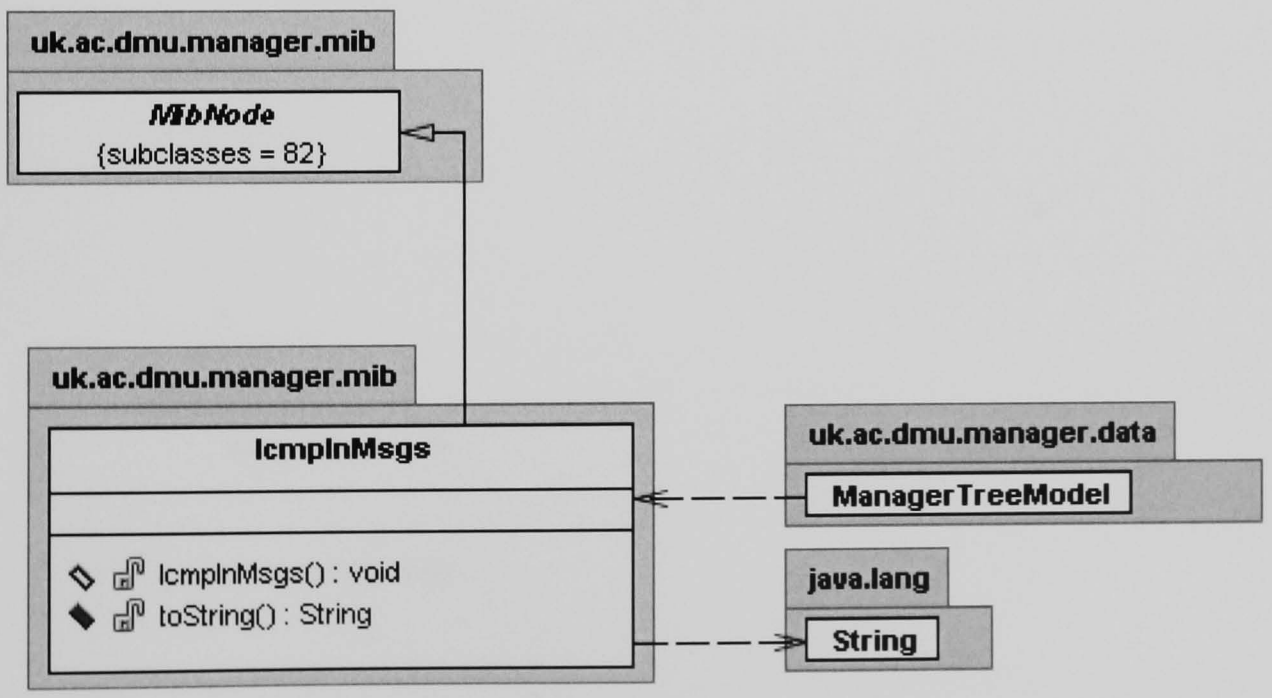
Class: IcmpInEchos



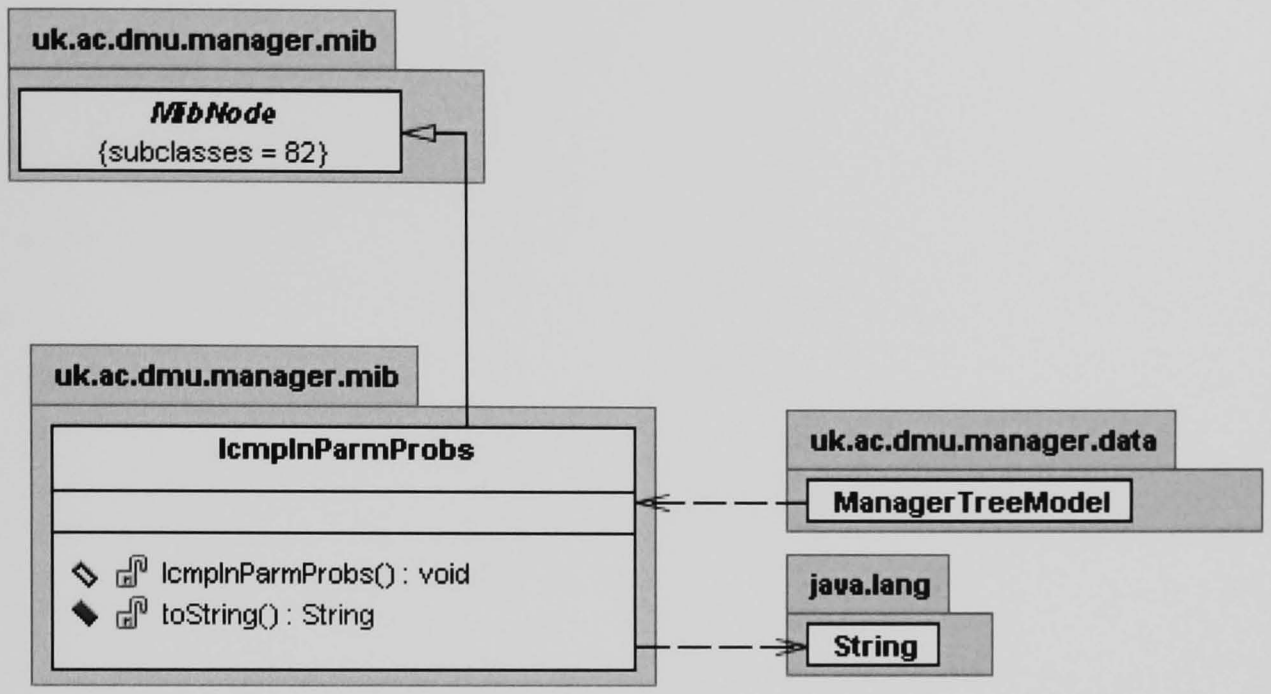
Class IcmpInErrors



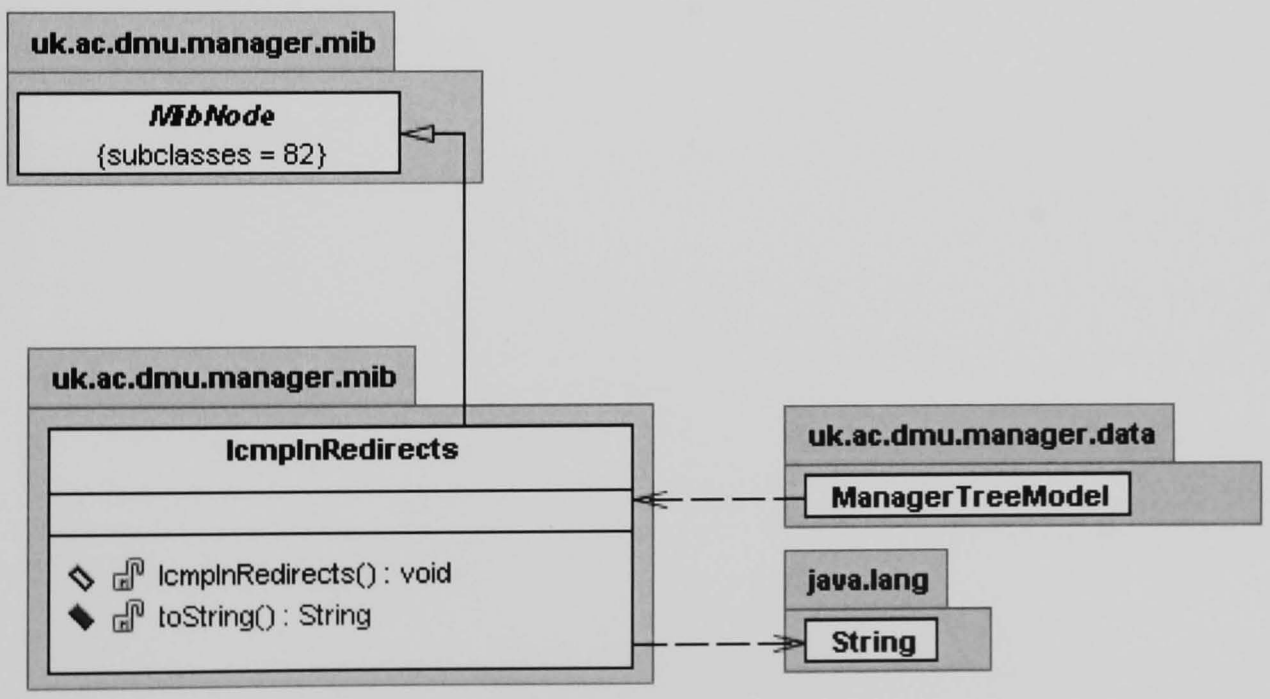
Class: IcmpInMsgs



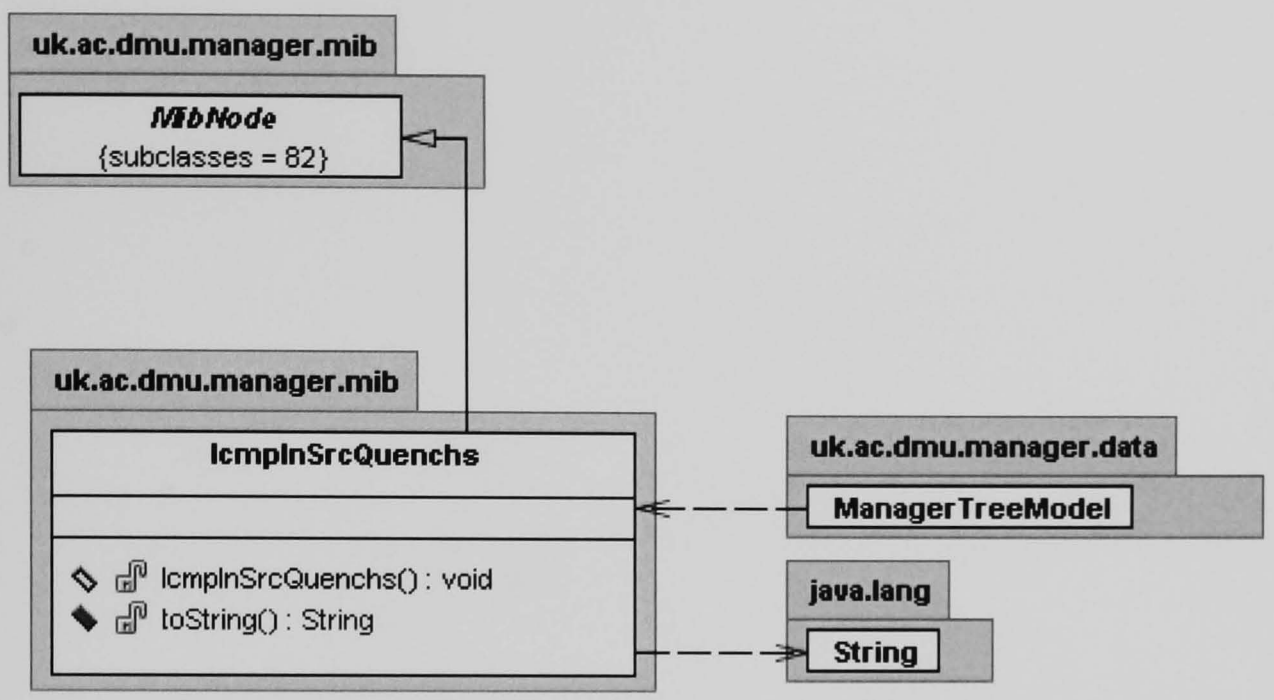
Class: IcmpInParmProbs



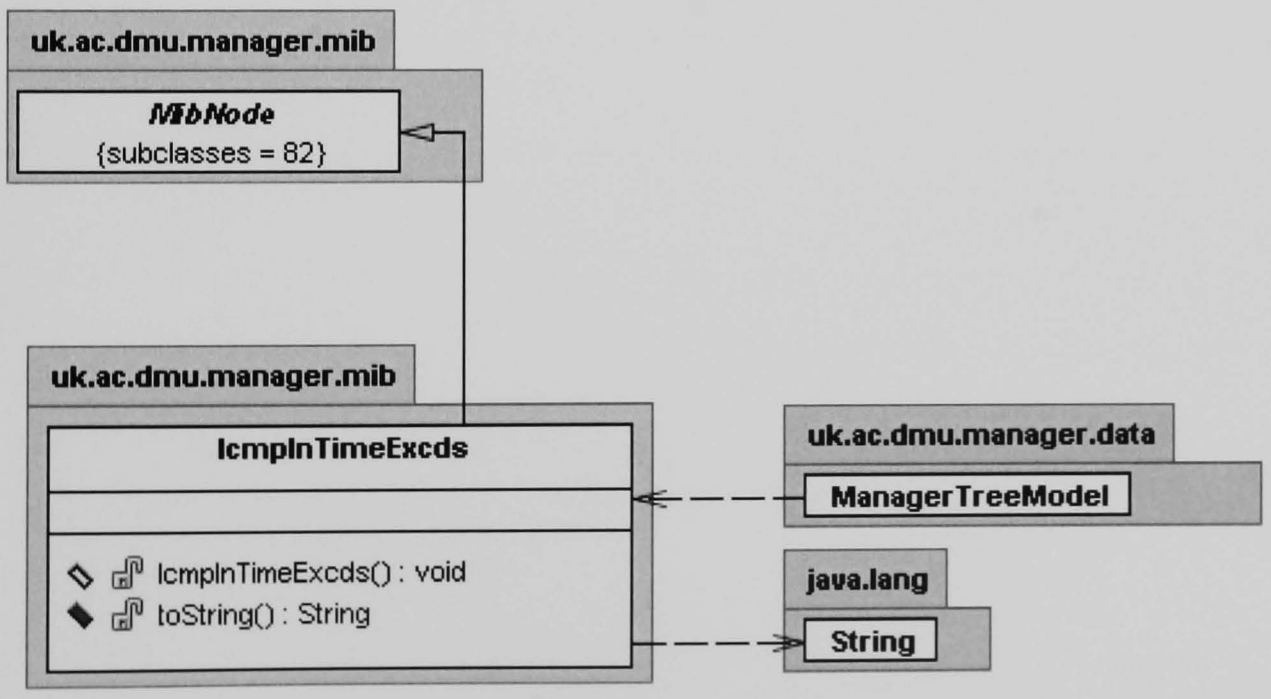
Class:IcmpInRedirects



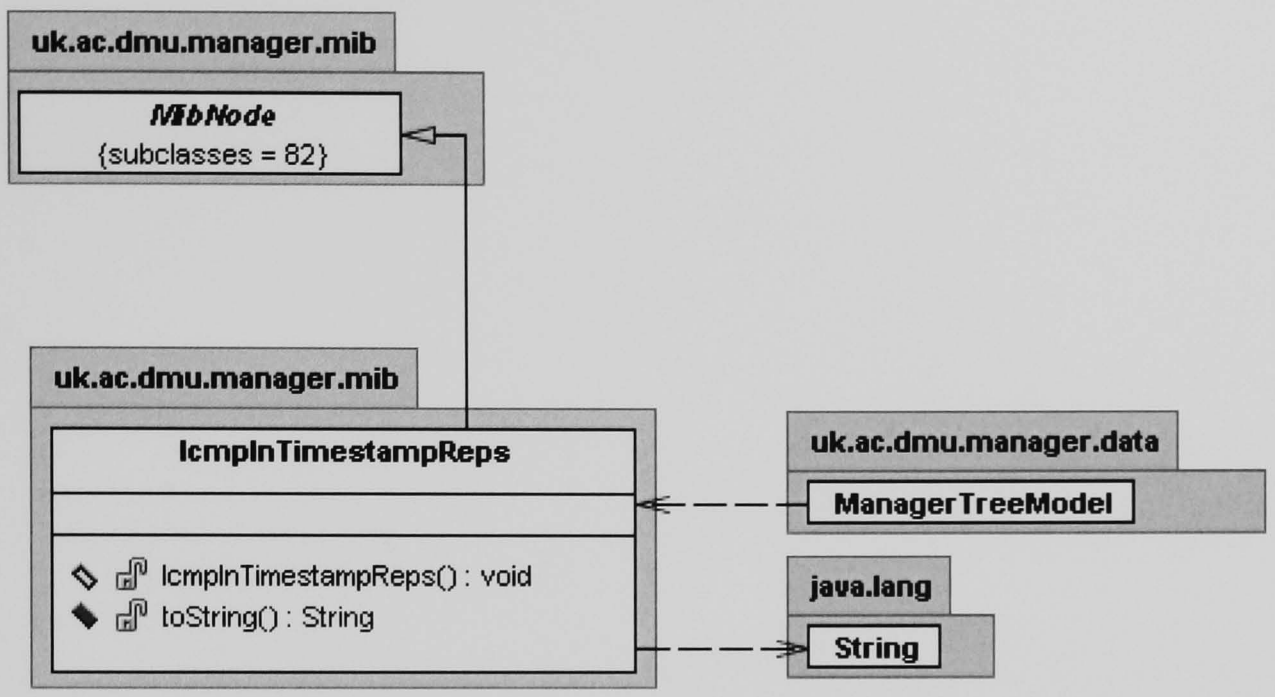
Class:IcmpInSrcQuenchs



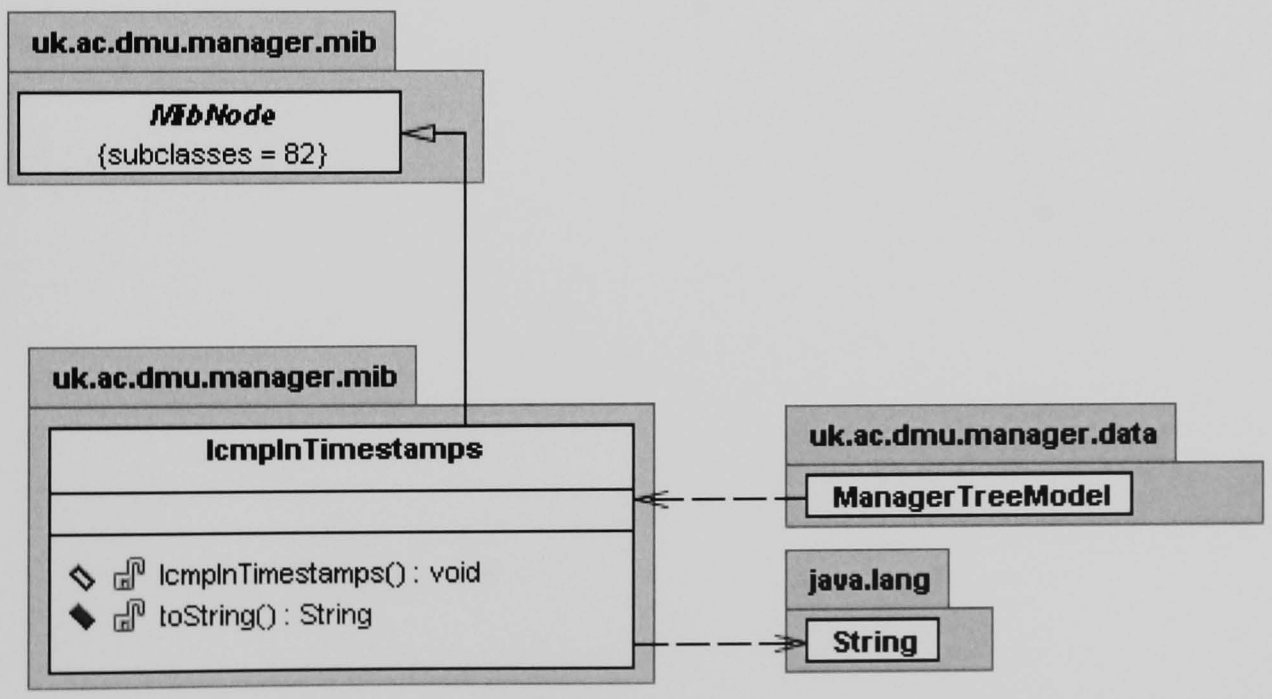
Class: IcmpInTimeExcds



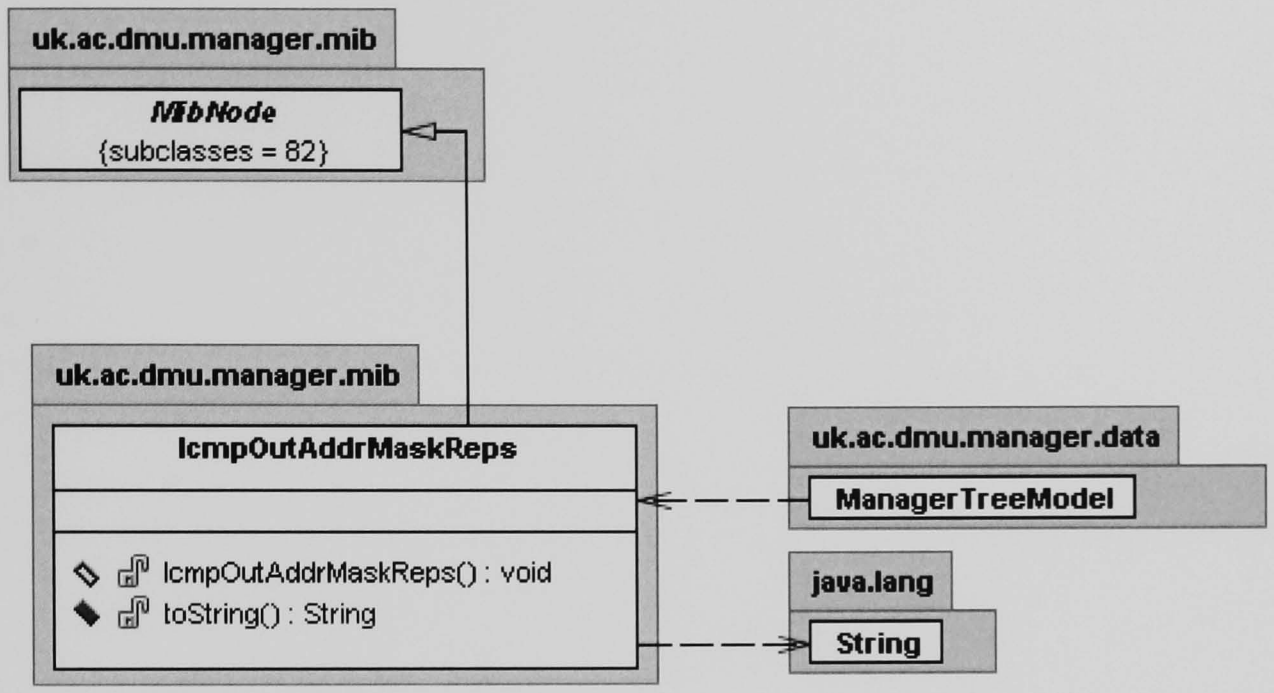
Class: IcmpInTimestampReps



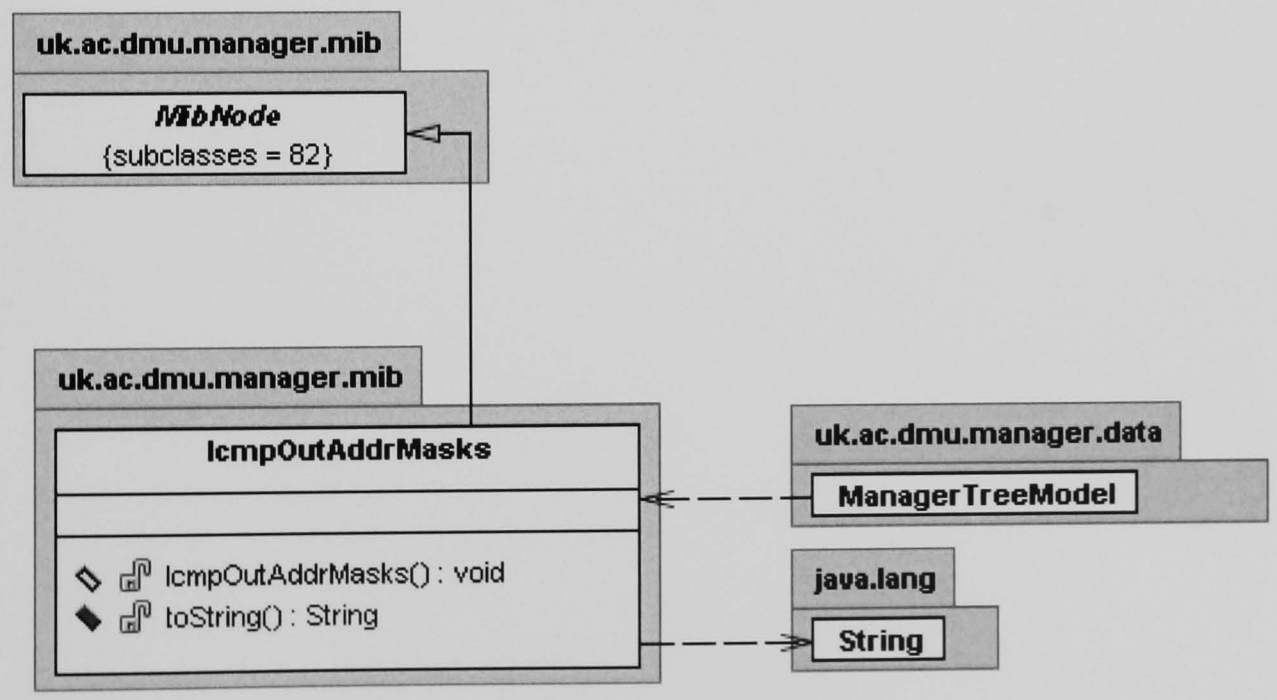
Class: IcmpInTimestamps



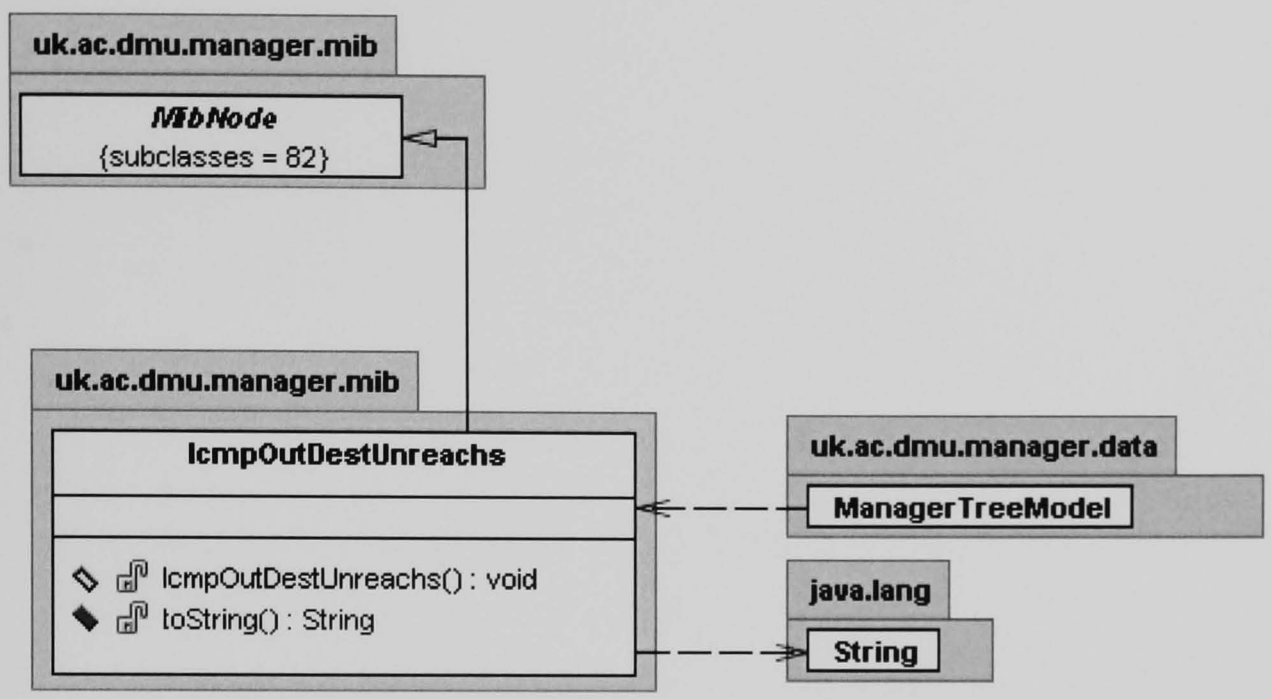
Class: IcmpOutAddrMaskReps



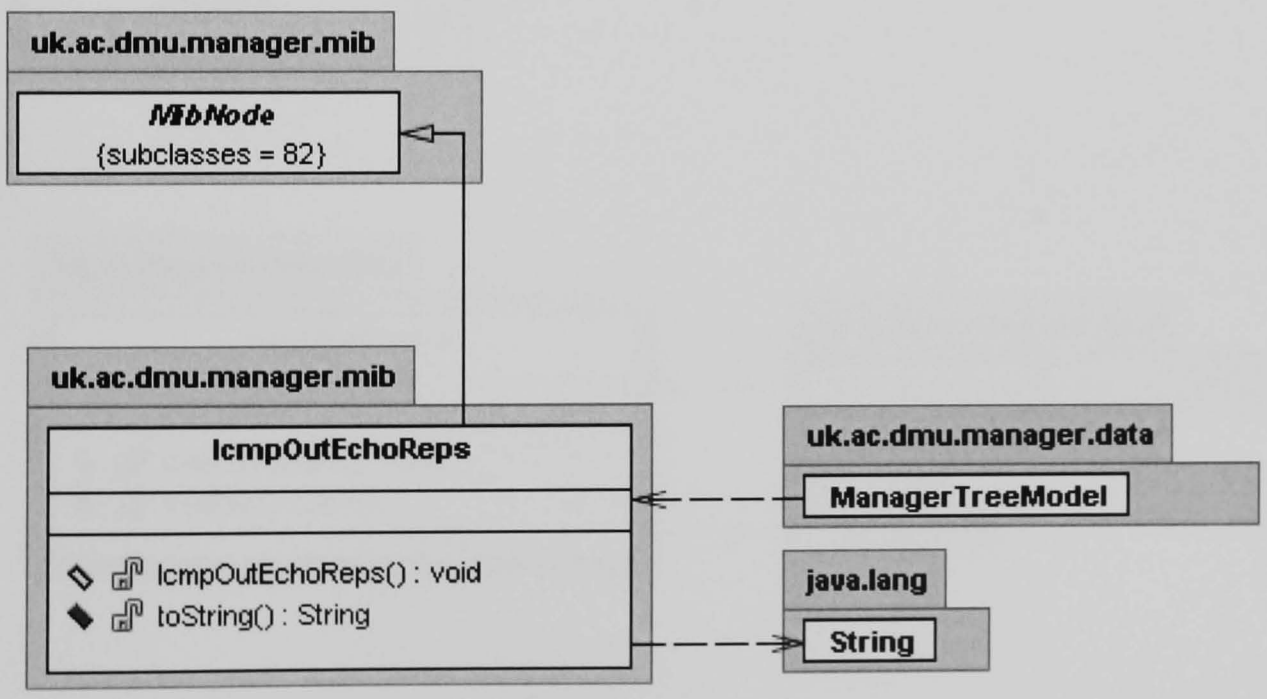
Class:IcmpOutAddrMasks



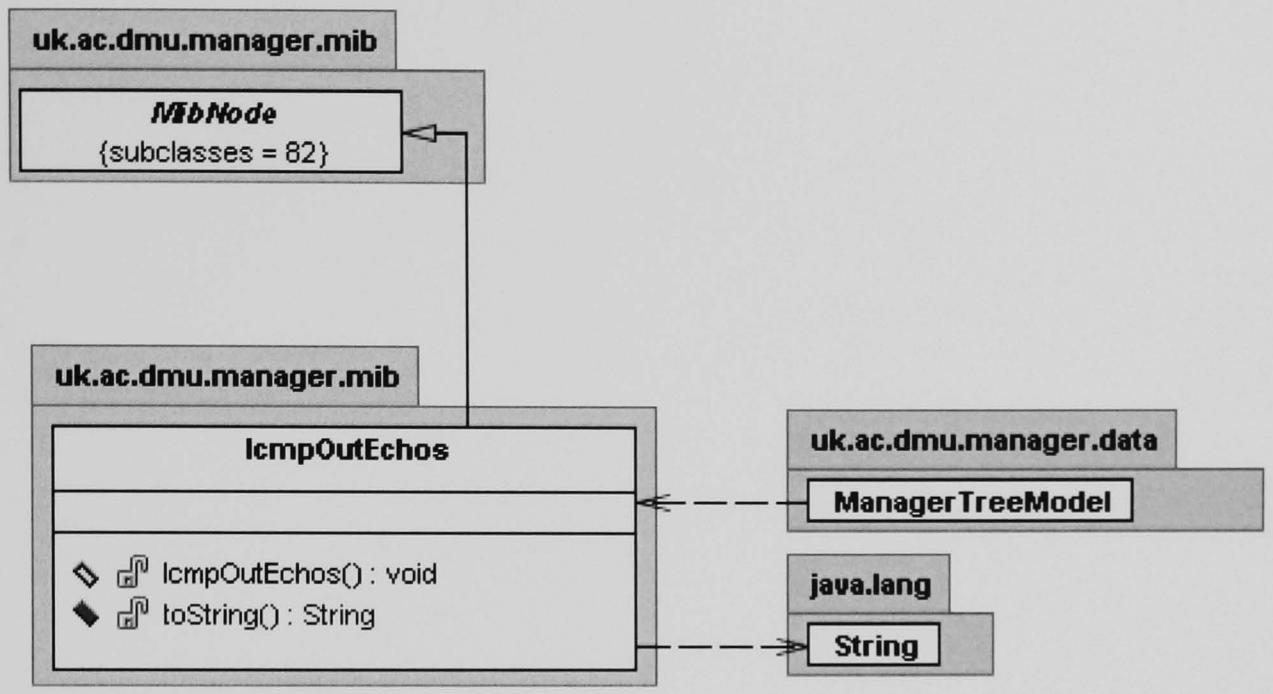
Class: IcmpOutDestUnreachs



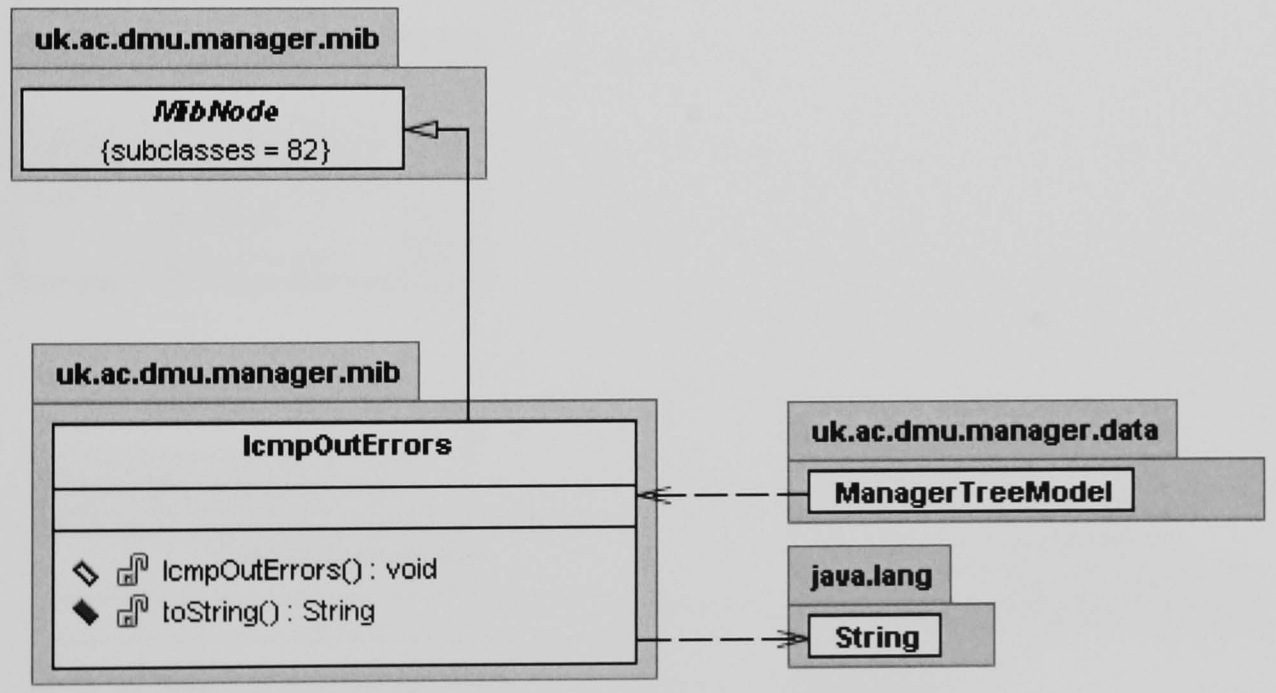
Class: IcmpOutEchoReps



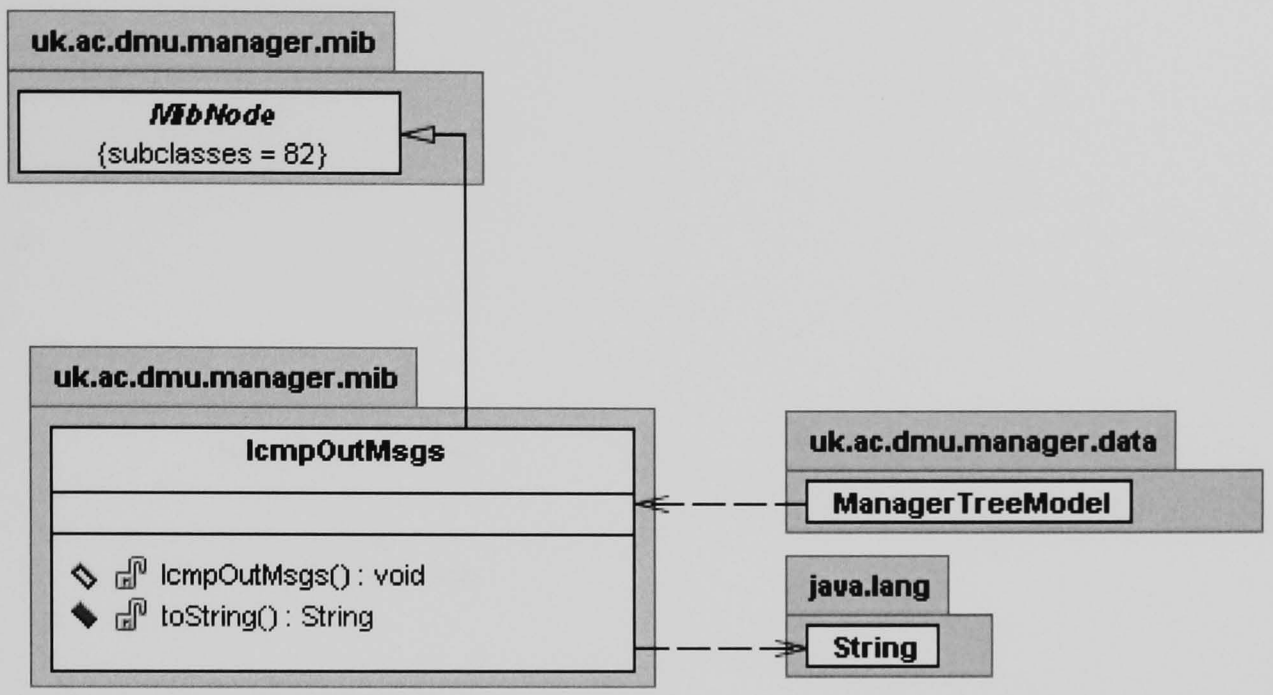
Class: IcmpOutEchos



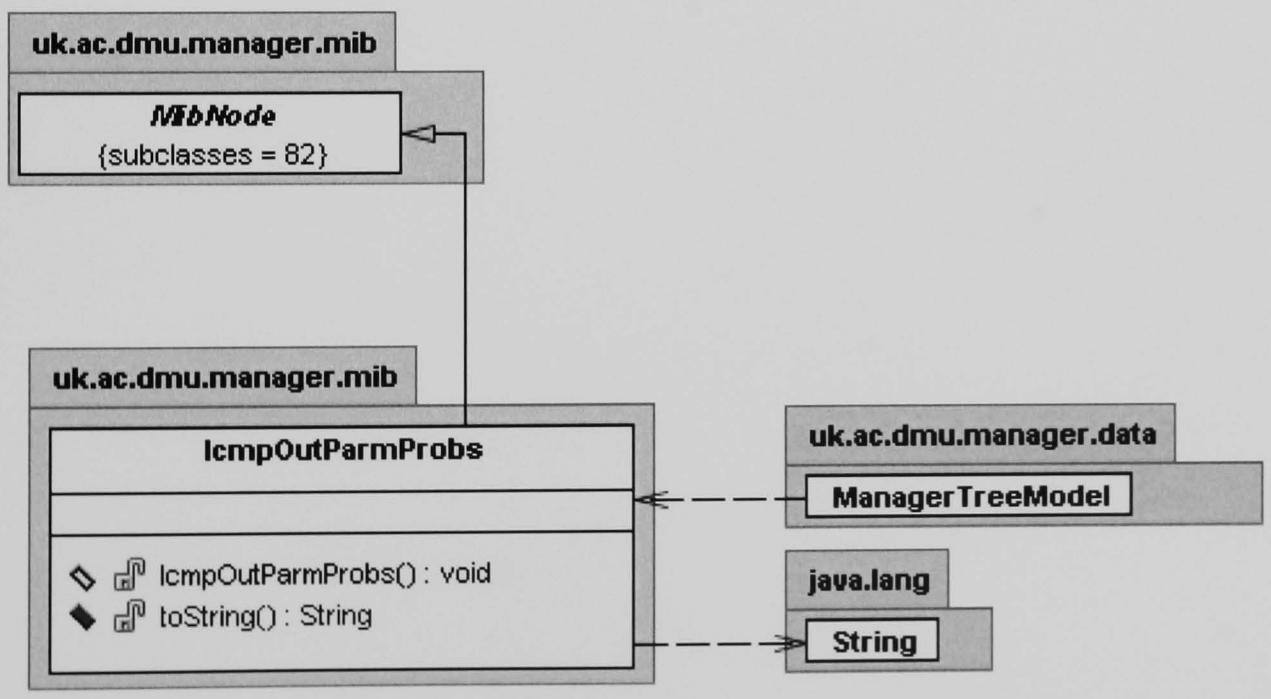
Class: IcmpOutErrors



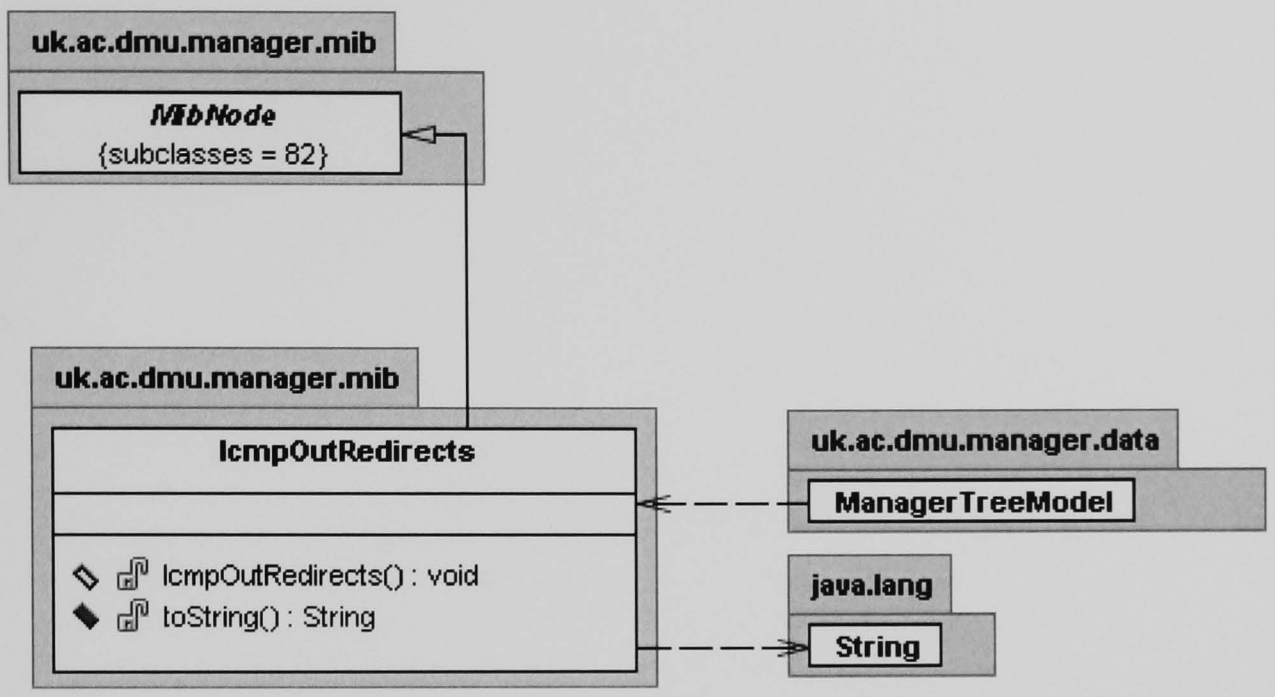
Class: IcmpOutMsgs



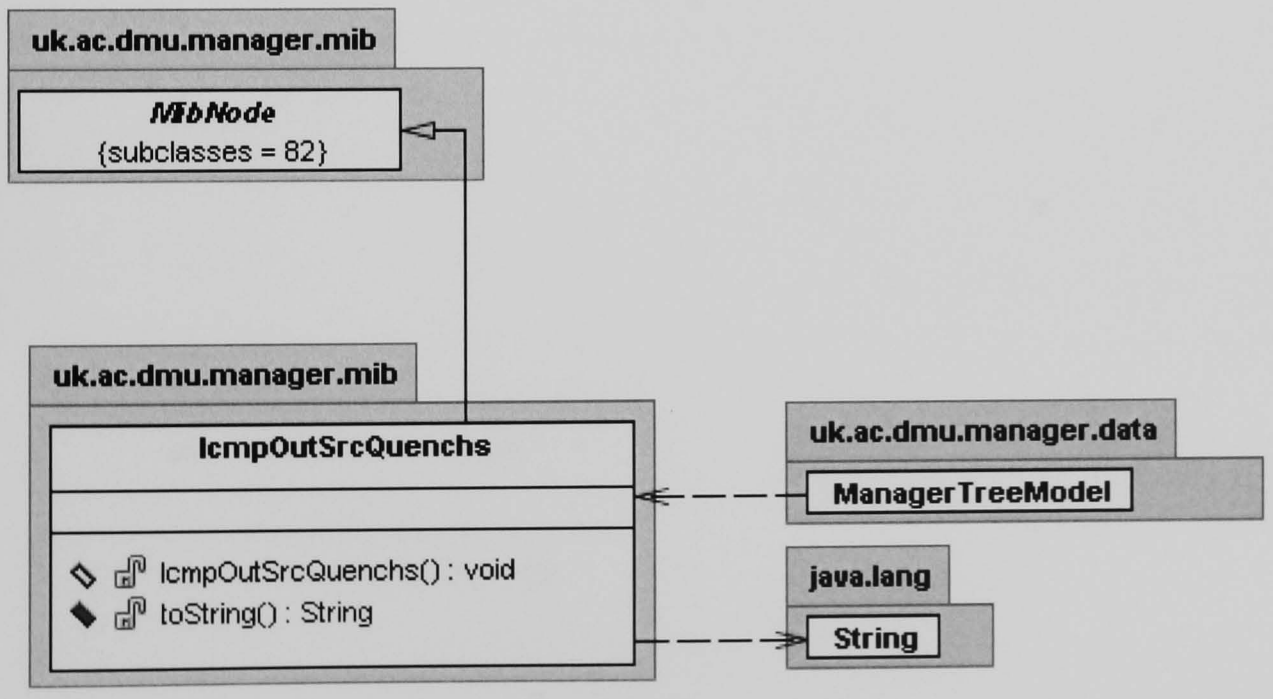
Class: IcmpOutParmProbs



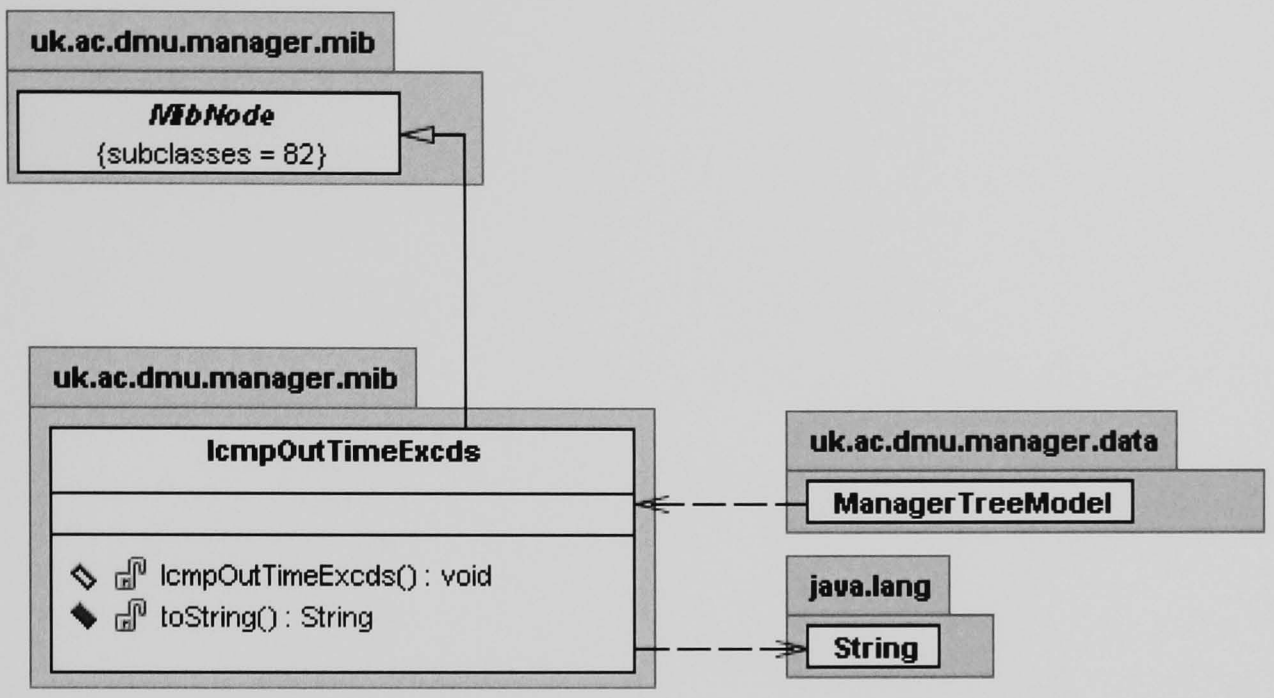
Class: IcmpOutRedirects



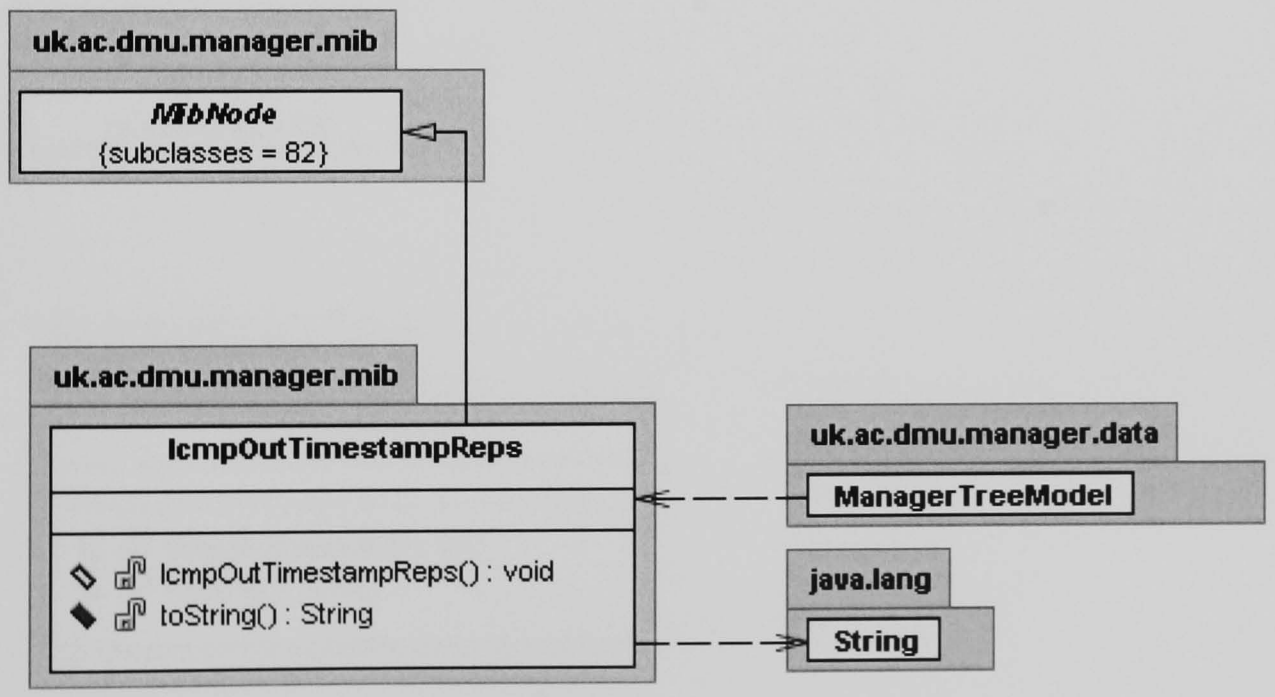
Class: IcmpOutSrcQuenchs



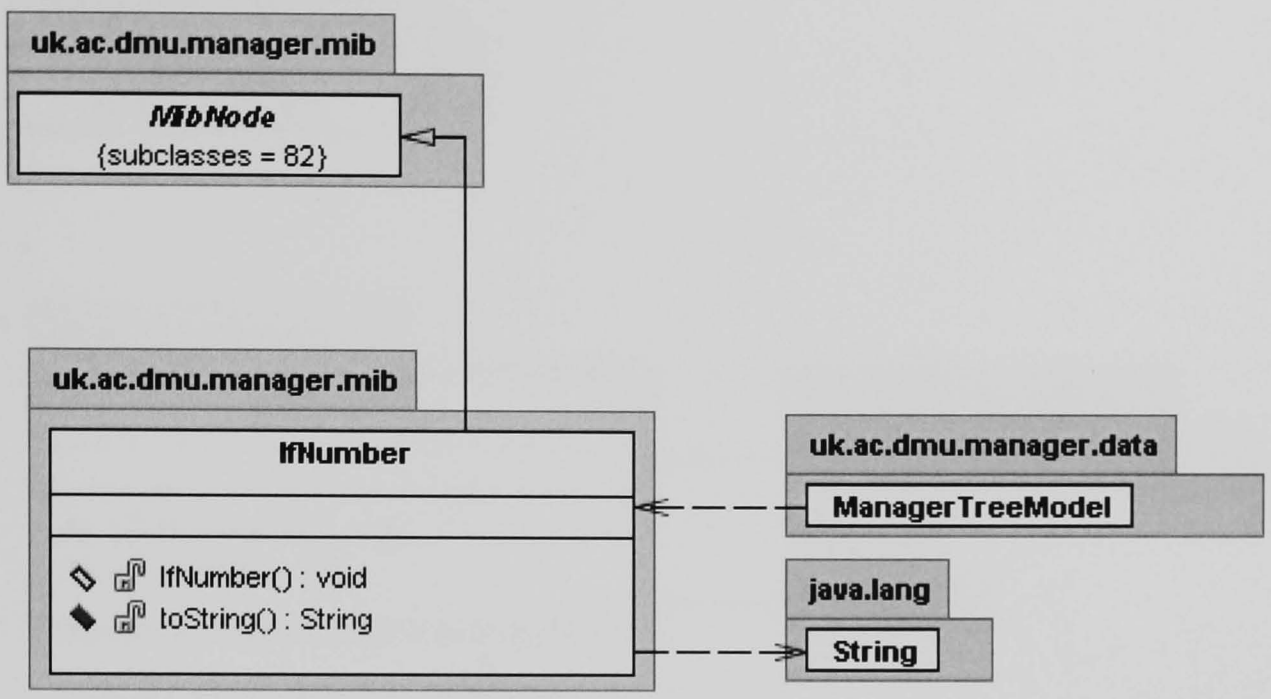
Class: IcmpOutTimeExcds



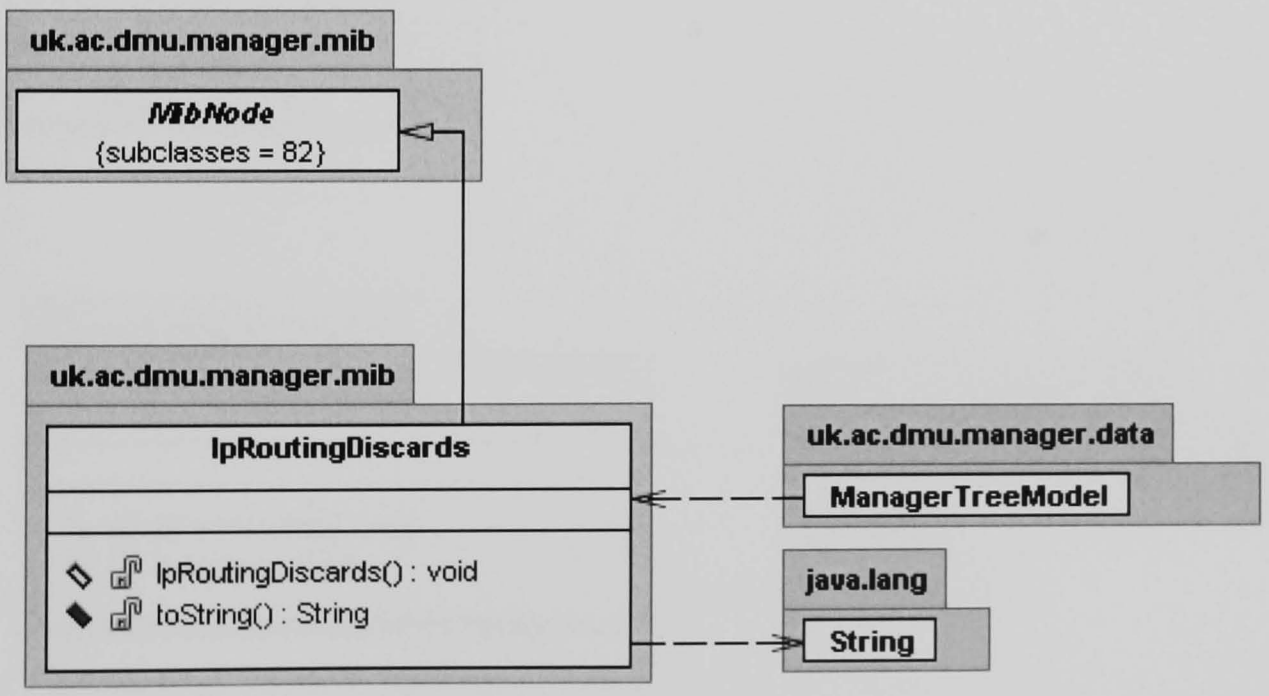
Class: IcmpOutTimestampReps



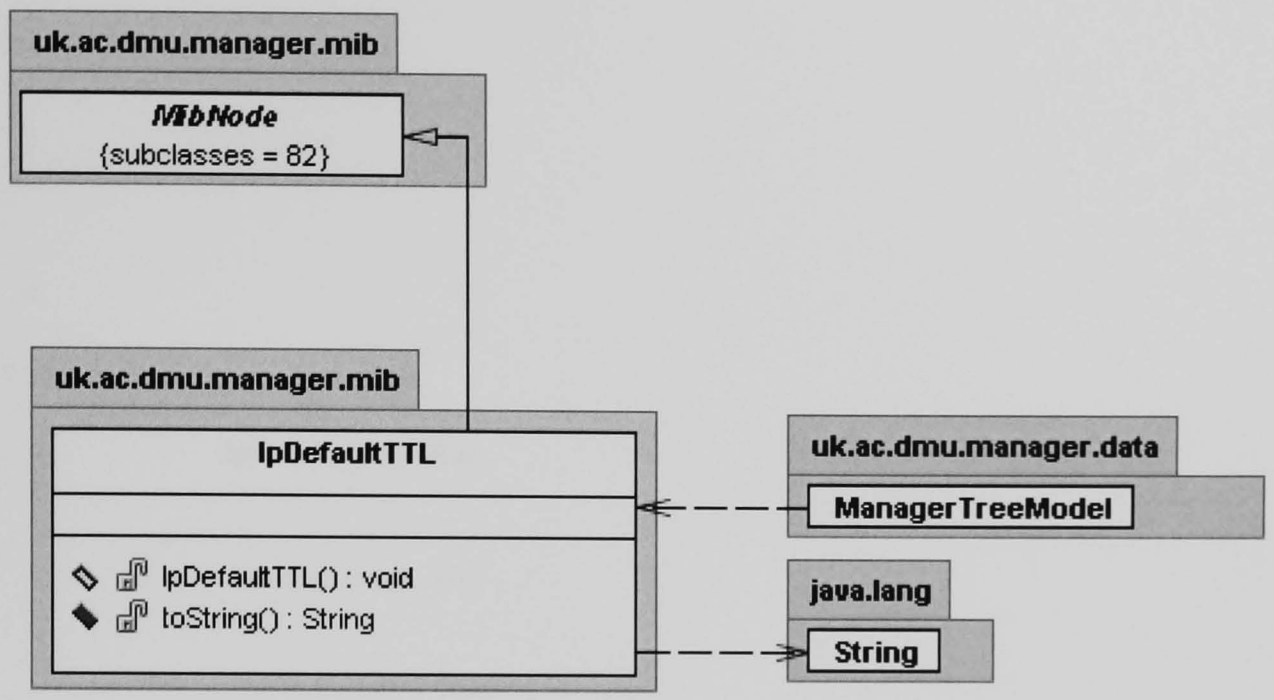
Class: IfNumber



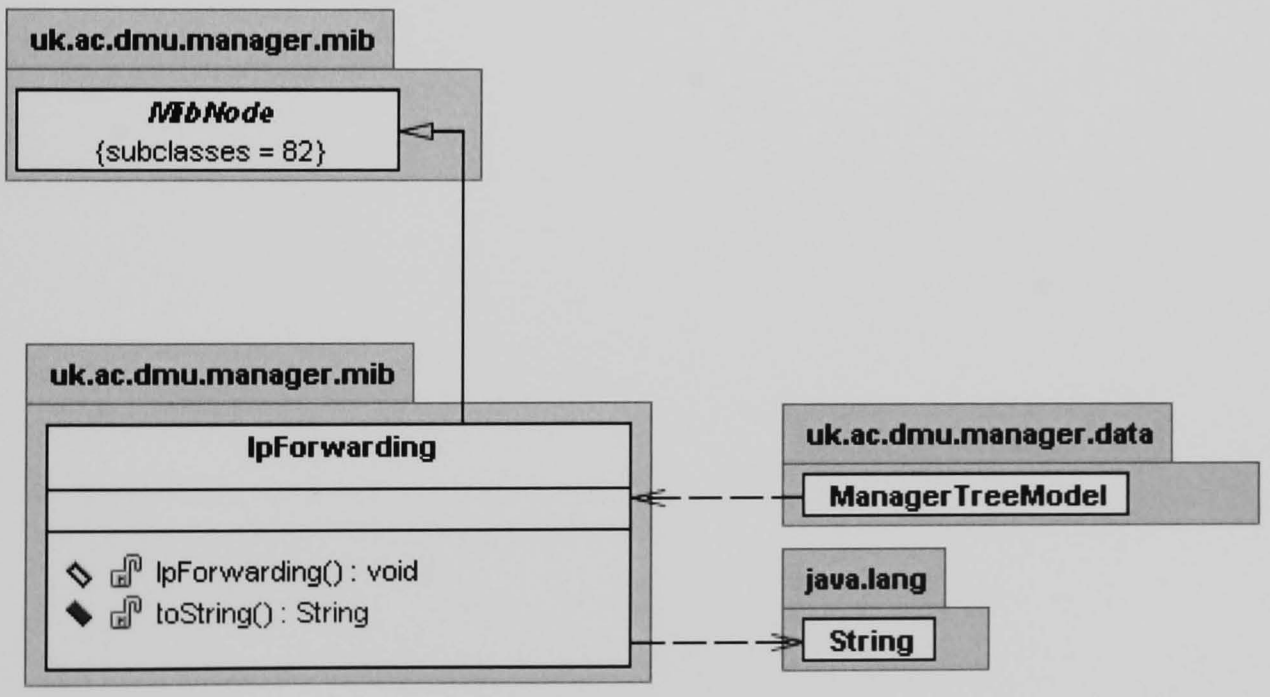
Class: IpRoutingDiscards



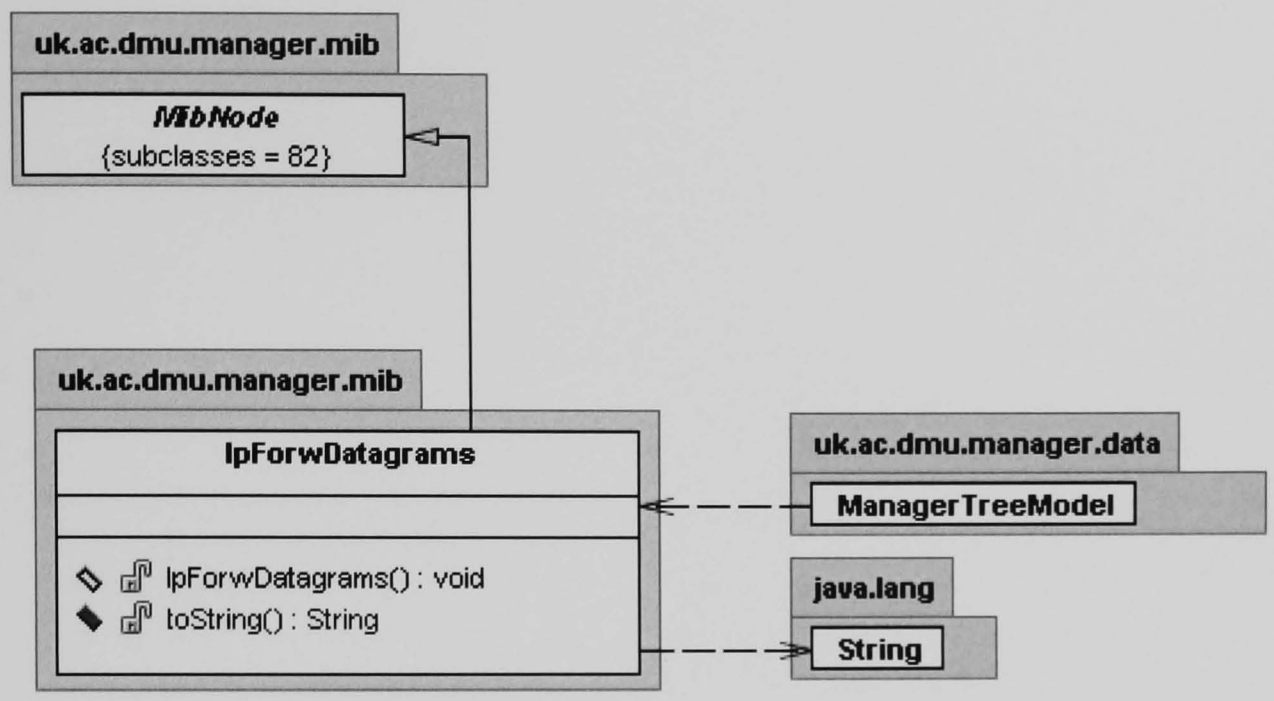
Class: IpDefaultTTL



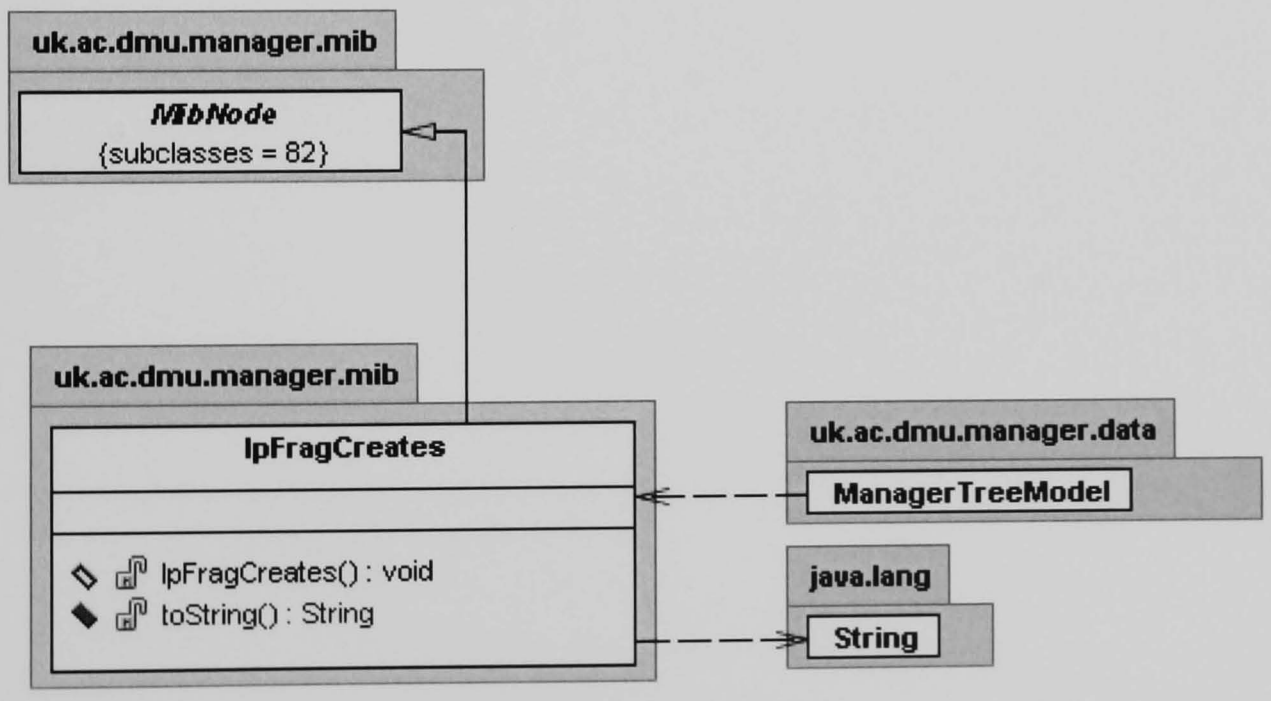
Class: IpForwarding



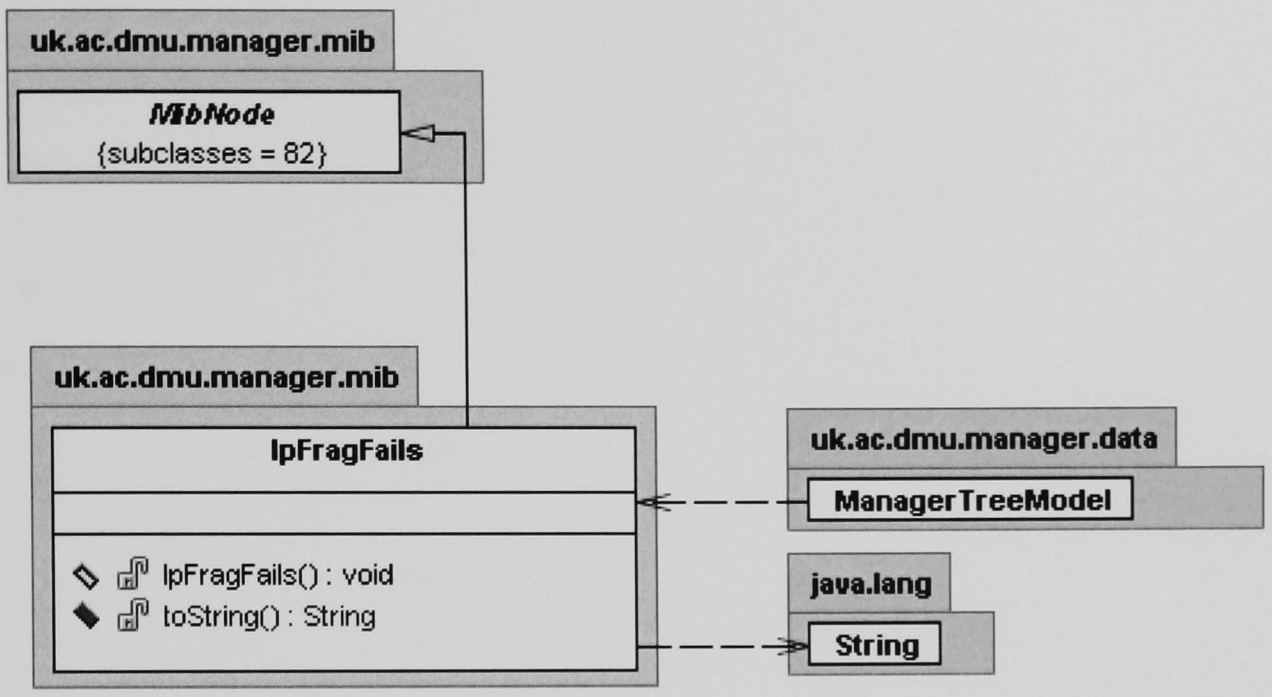
Class: IpForwDatagrams



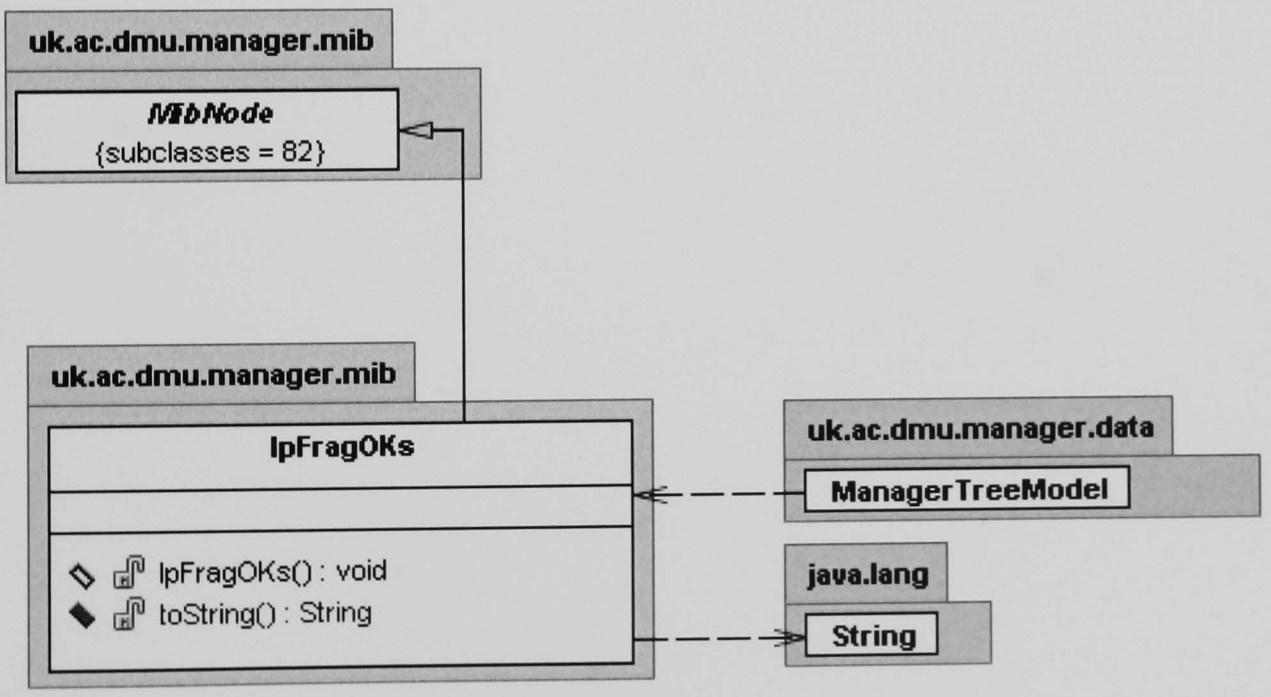
Class: IpFragCreates



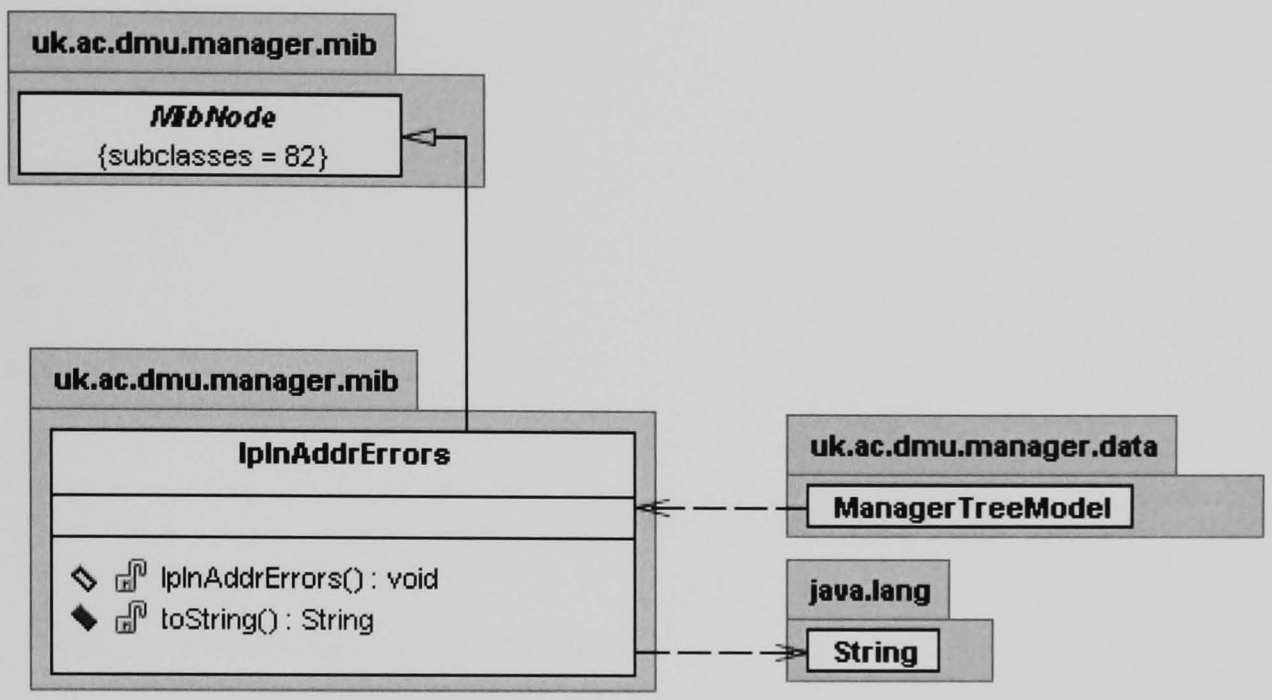
Class: IpFragFails



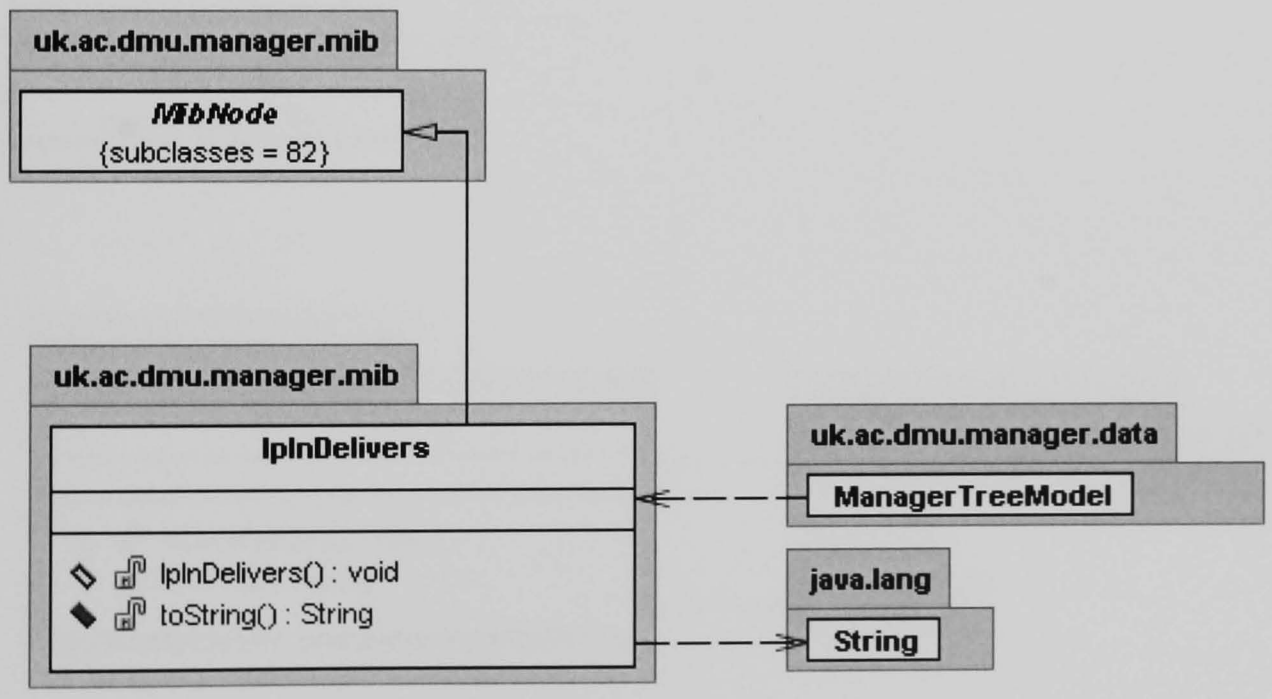
Class: IpFragsOKs



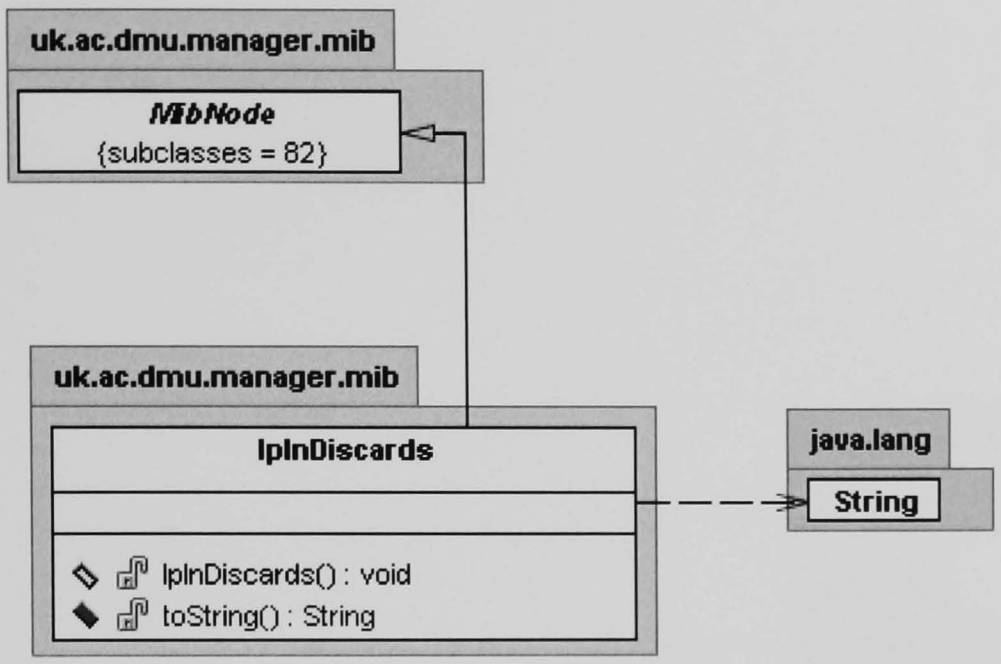
Class IpInAddrErrors



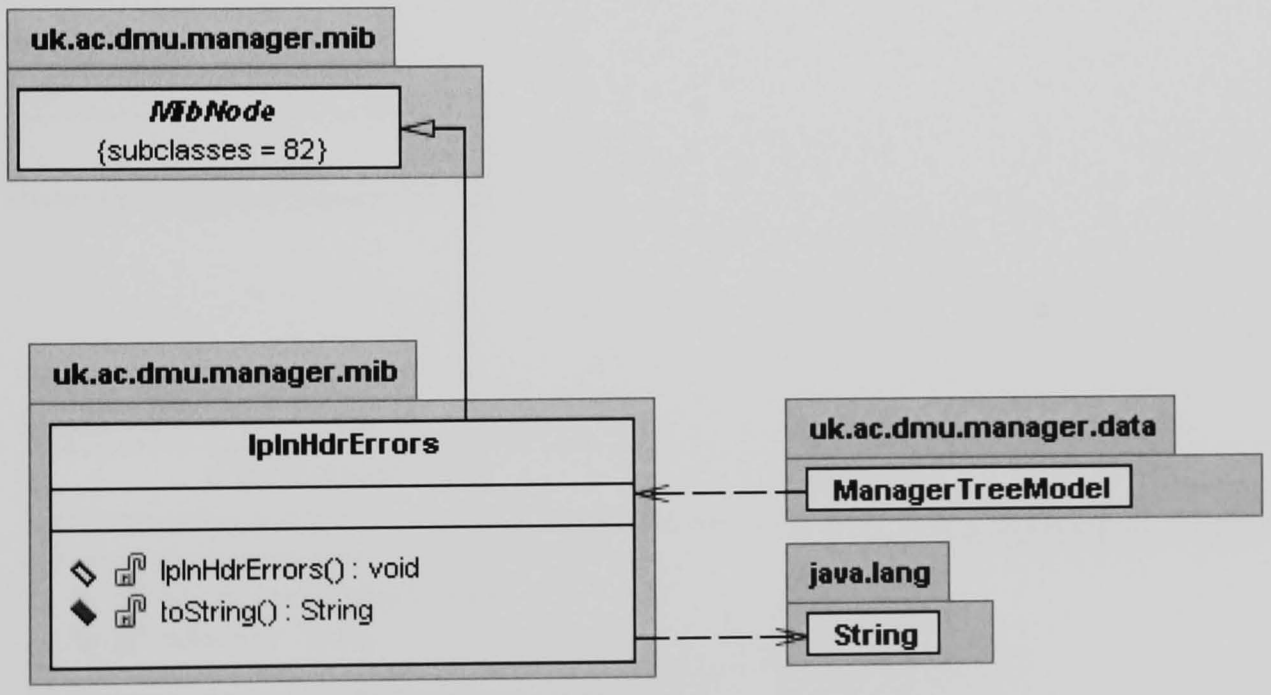
Class IpInDelivers



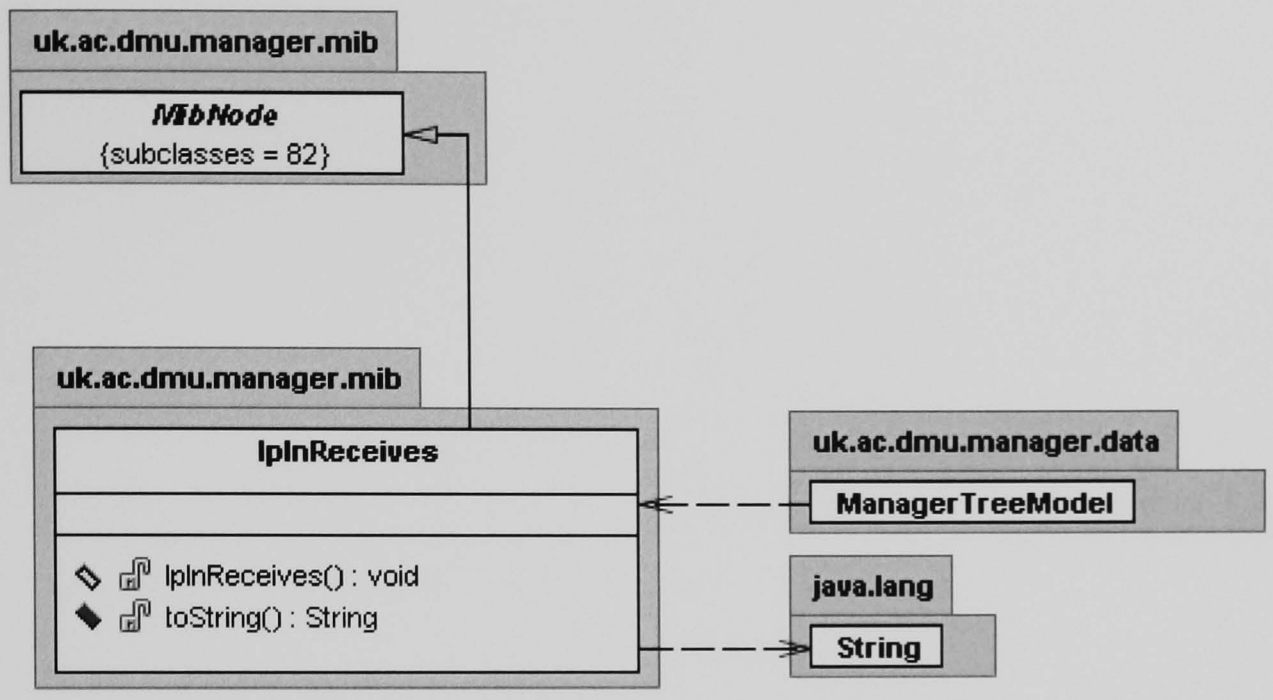
Class: IpInDiscards



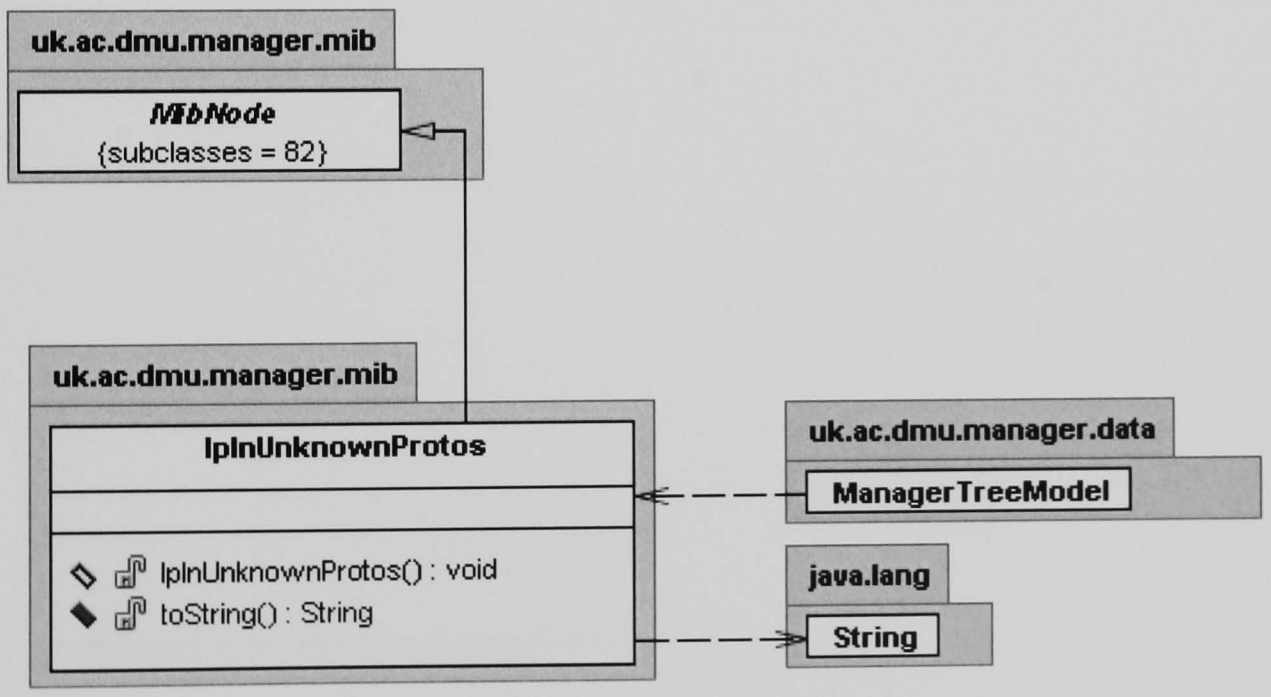
Class: IpInHdrErrors



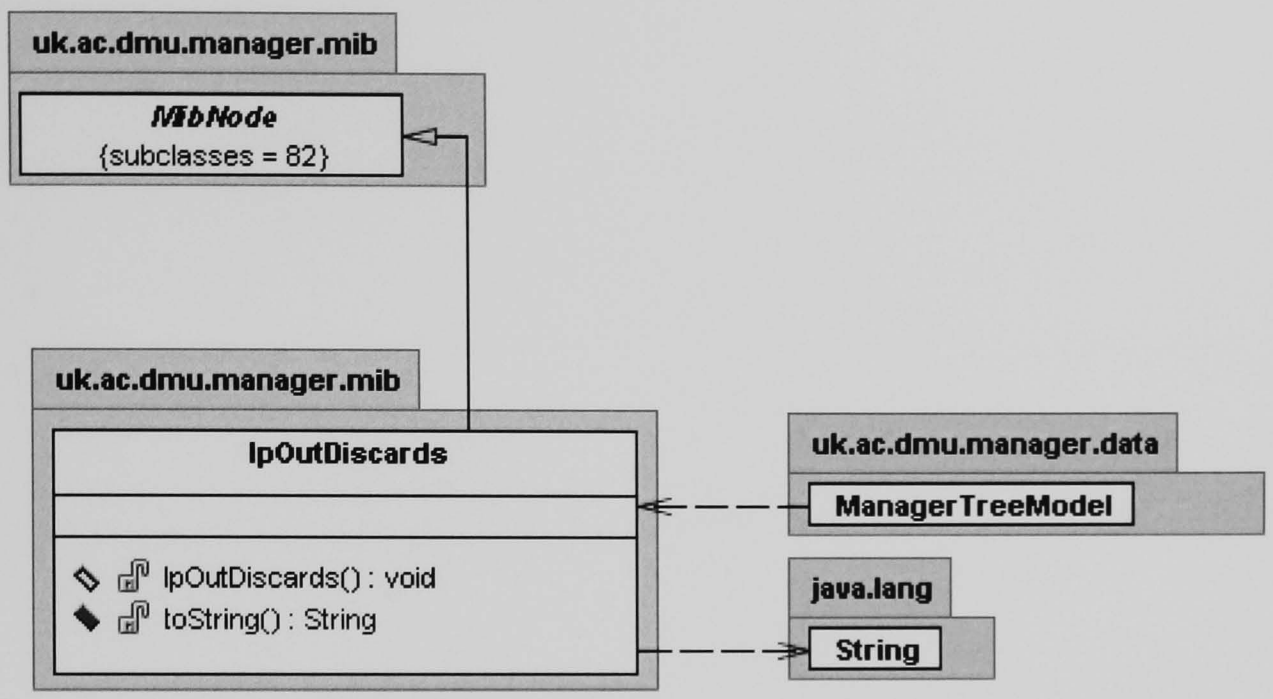
Class: IpInReceives



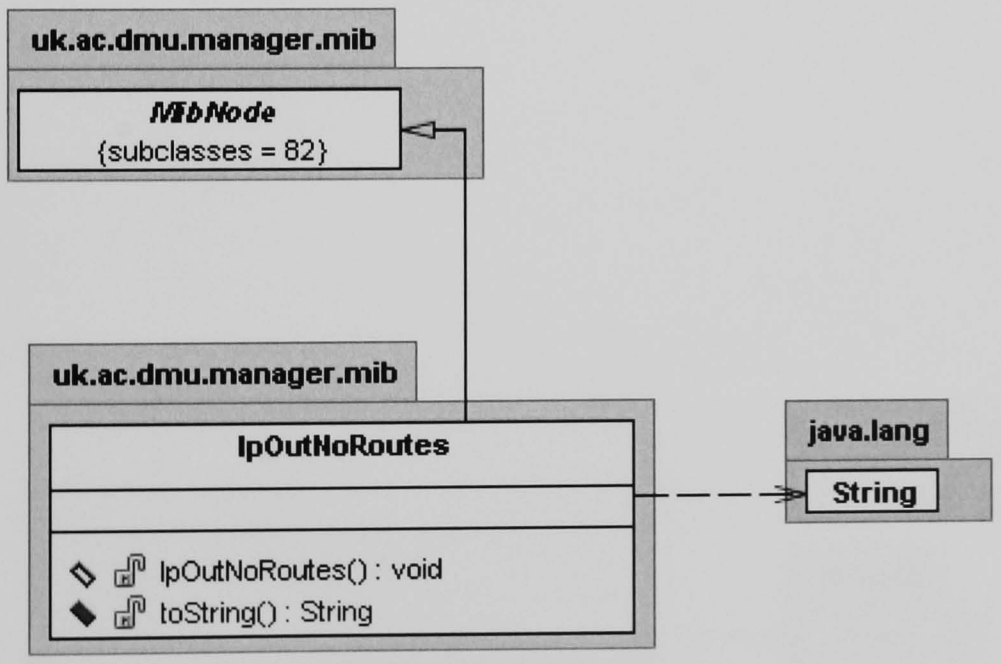
Class: IpInUnknownProtos



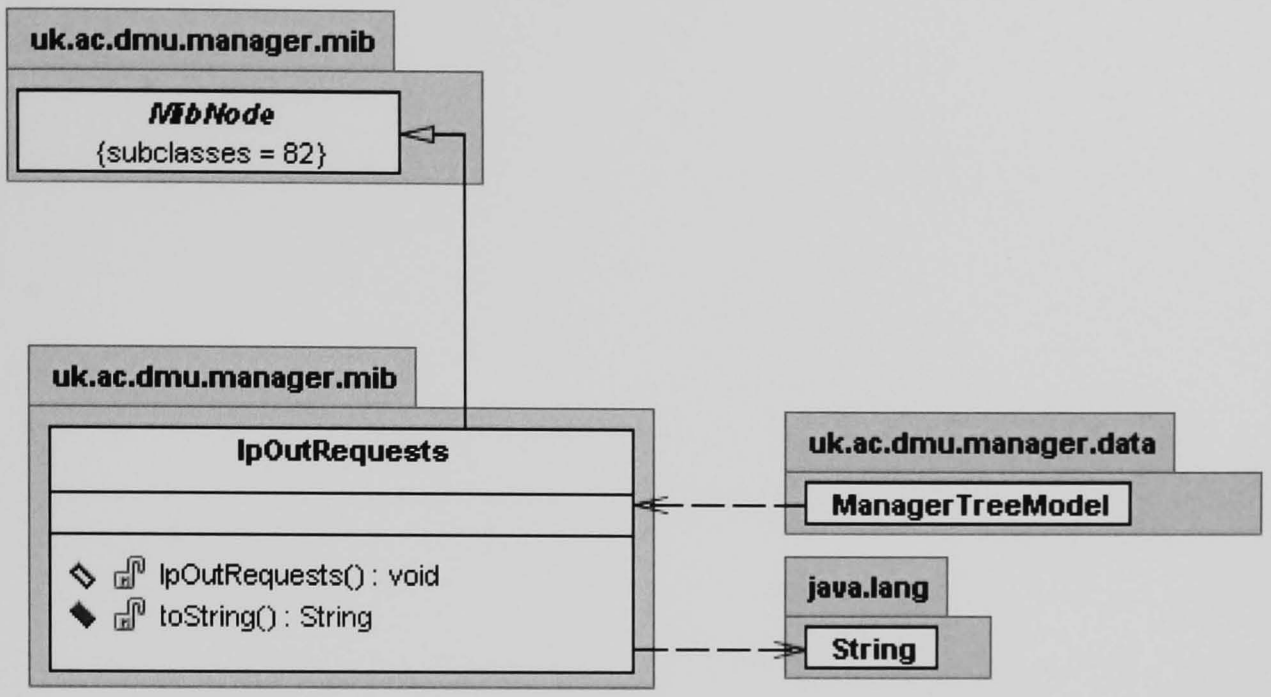
Class: IpOutDiscards



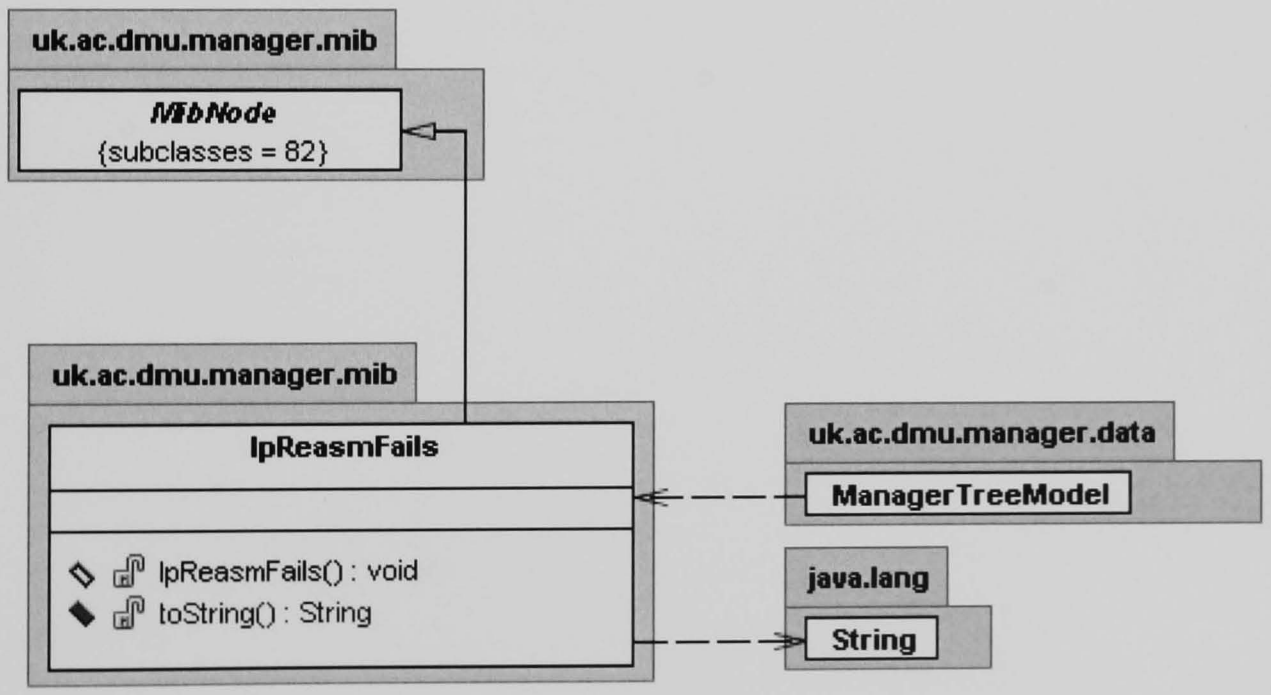
Class IpOutNoRoutes



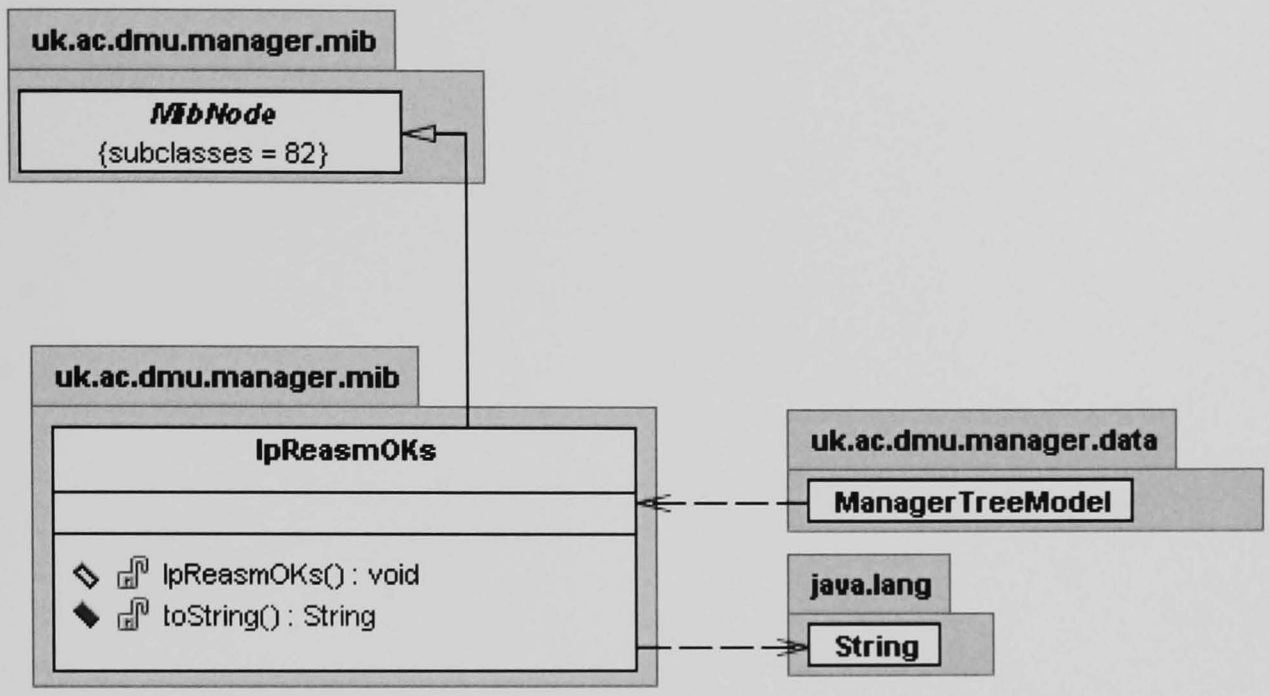
Class: IpOutRequests



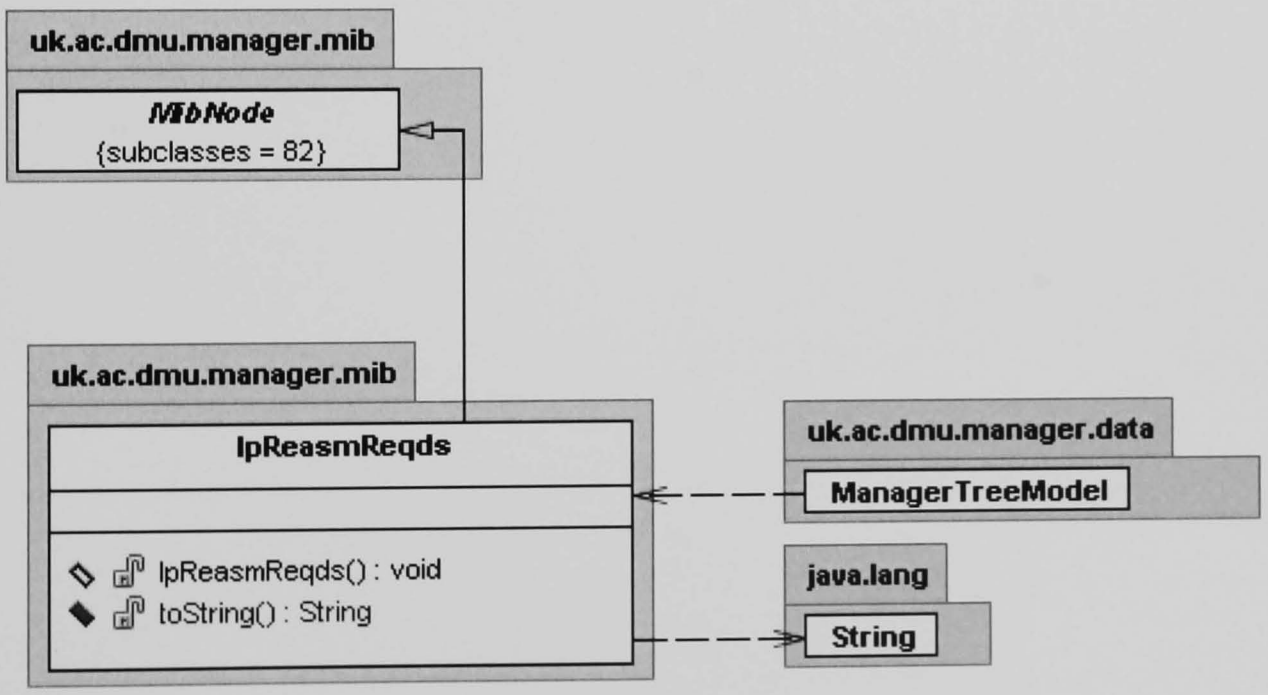
Class: IpReasmFails



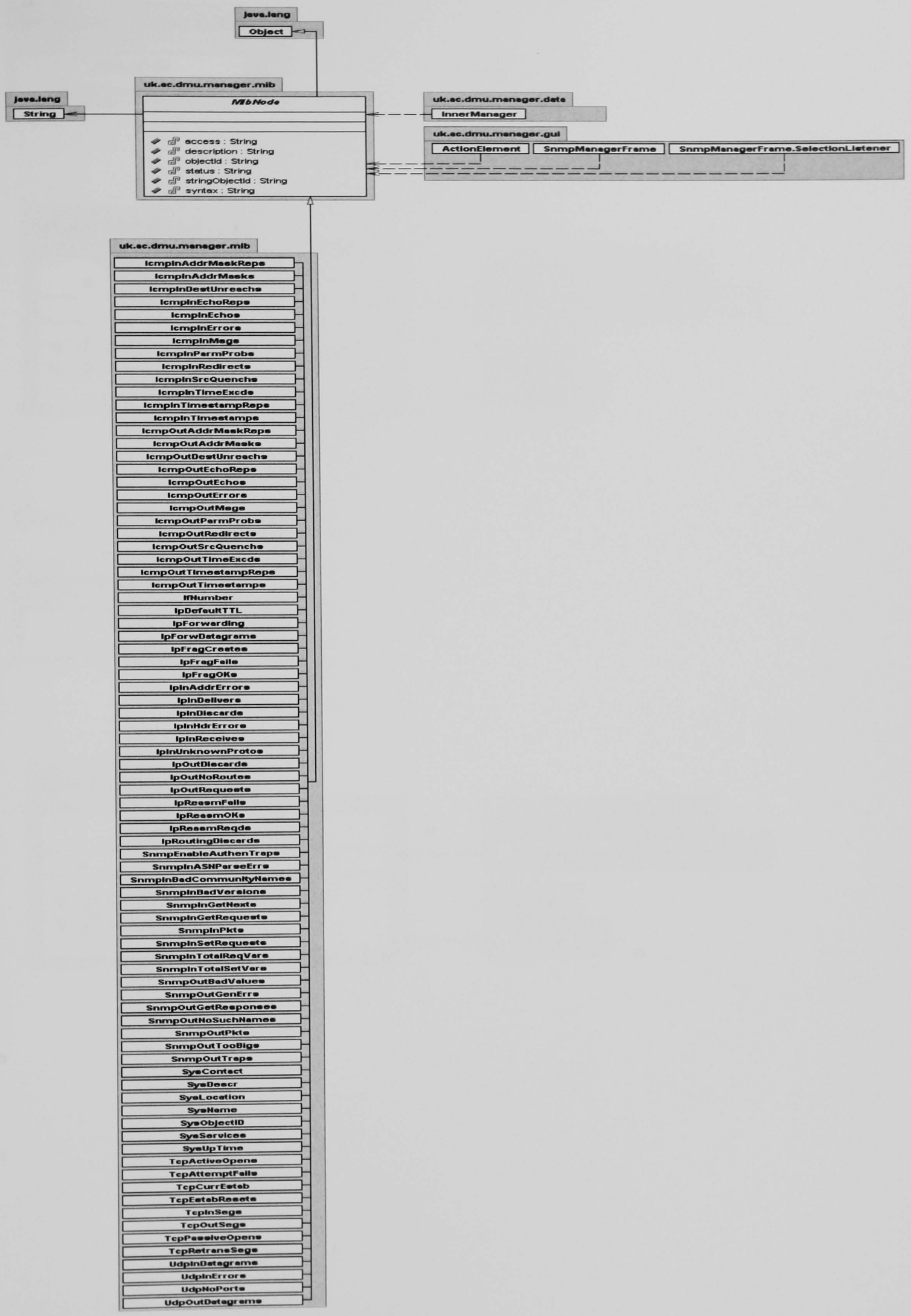
Class: IpReasmOKs



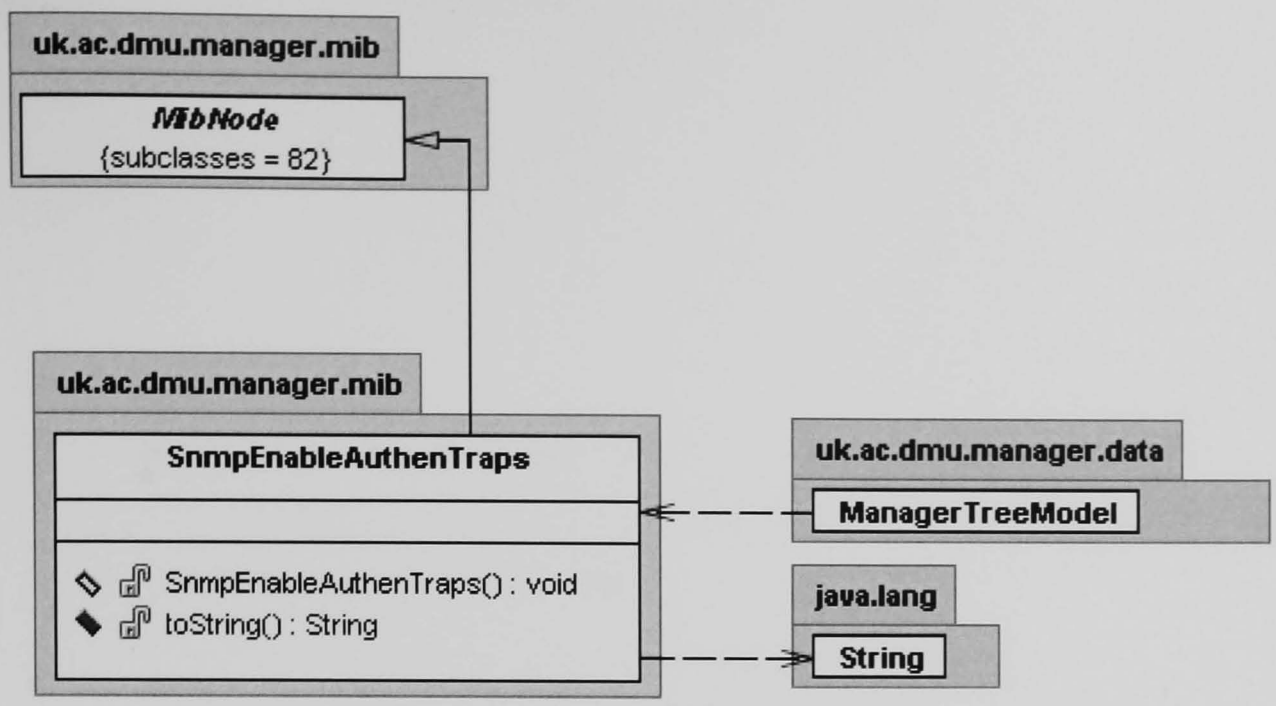
Class: IpReasmReqds



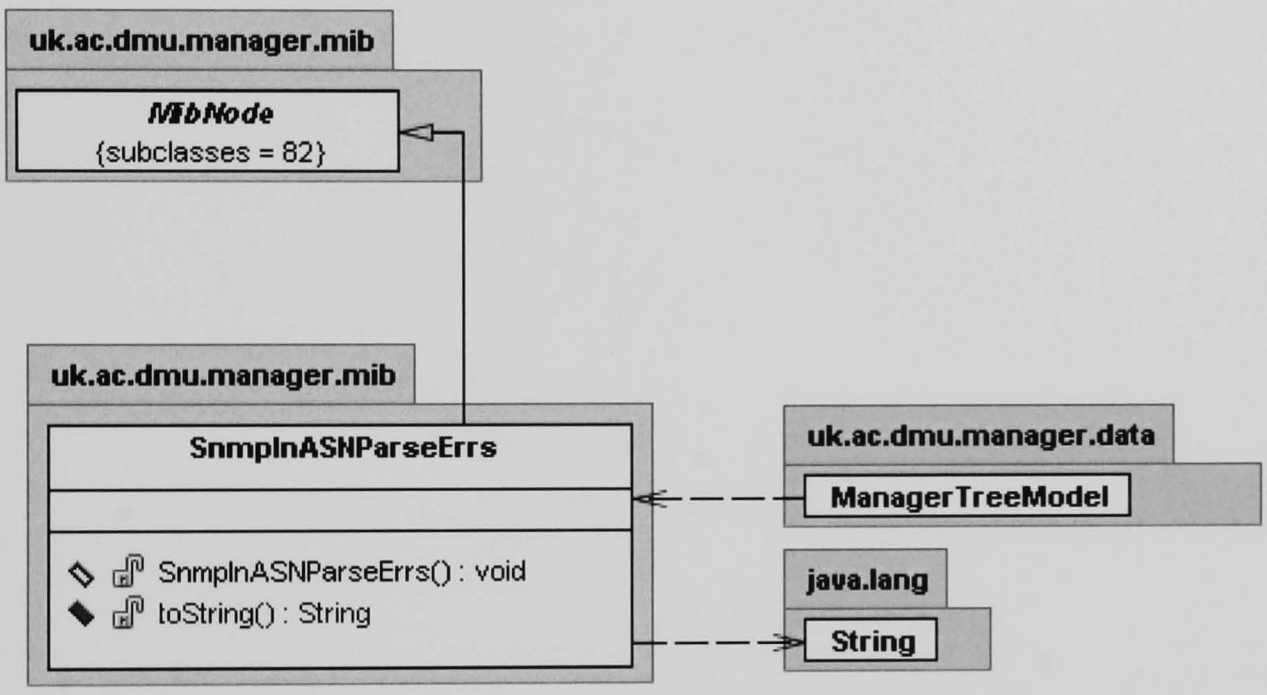
Class: MibNode



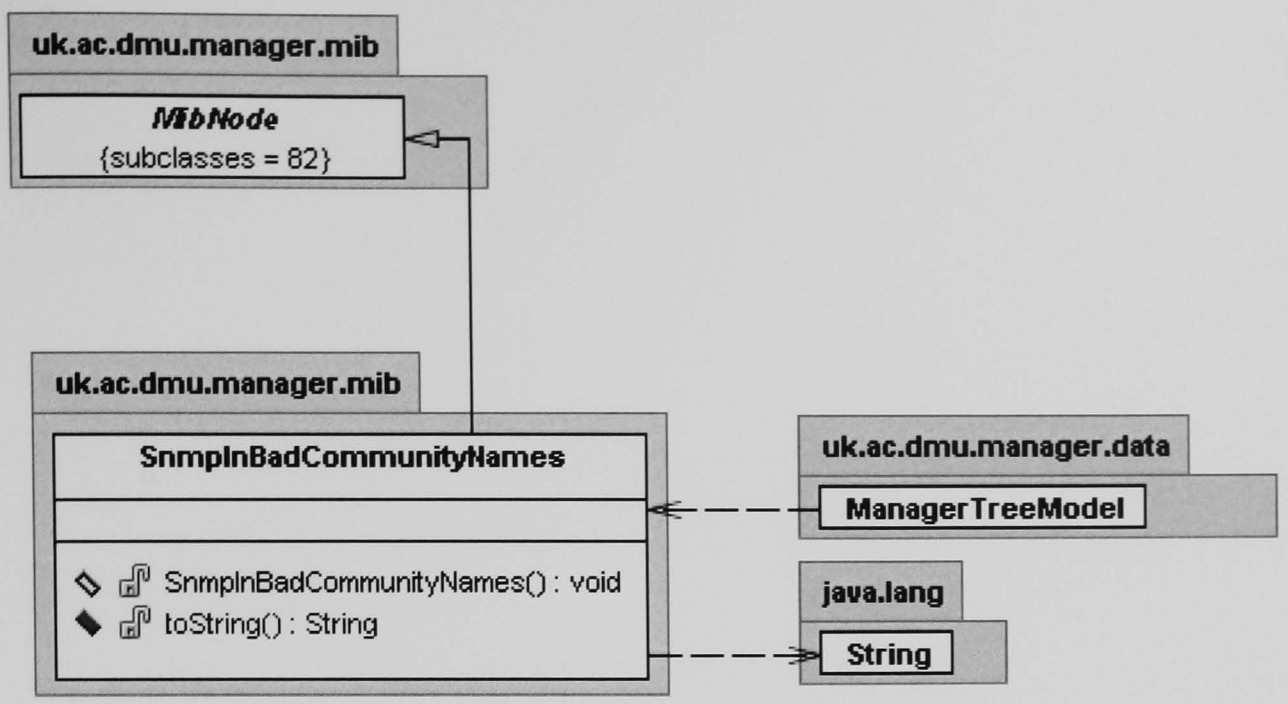
Class: SnmpEnableAuthenTraps



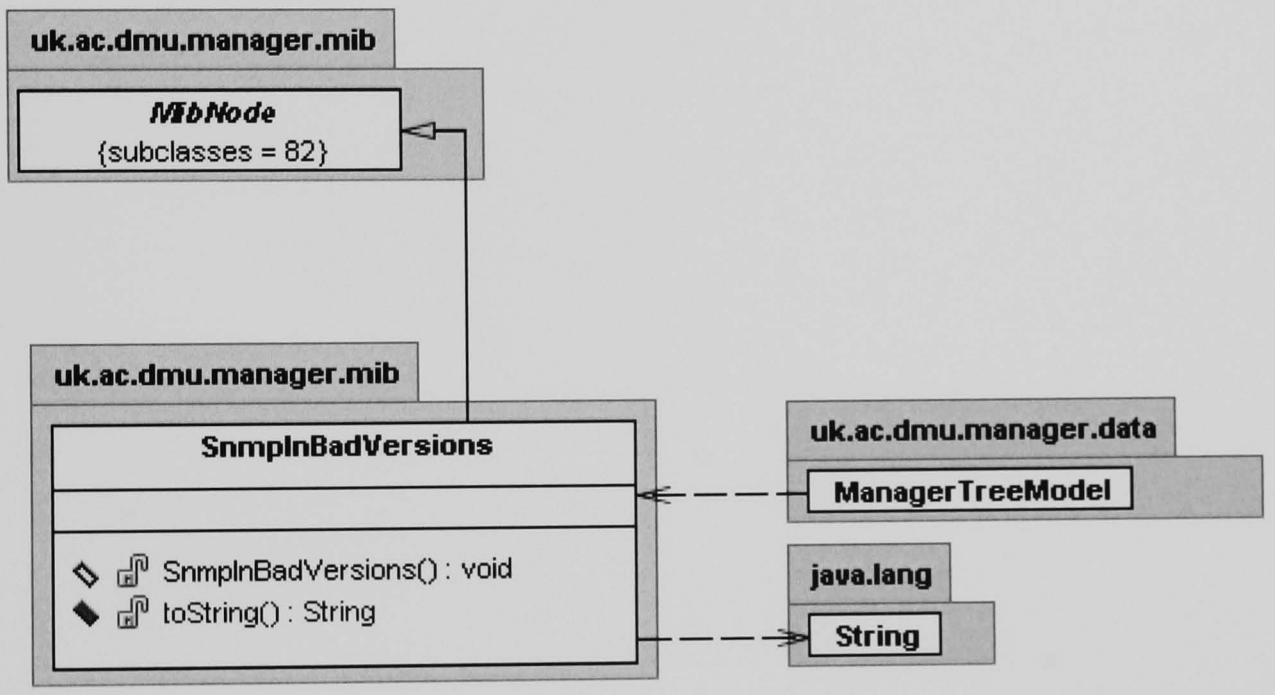
Class: SnmpInASNParseErrs



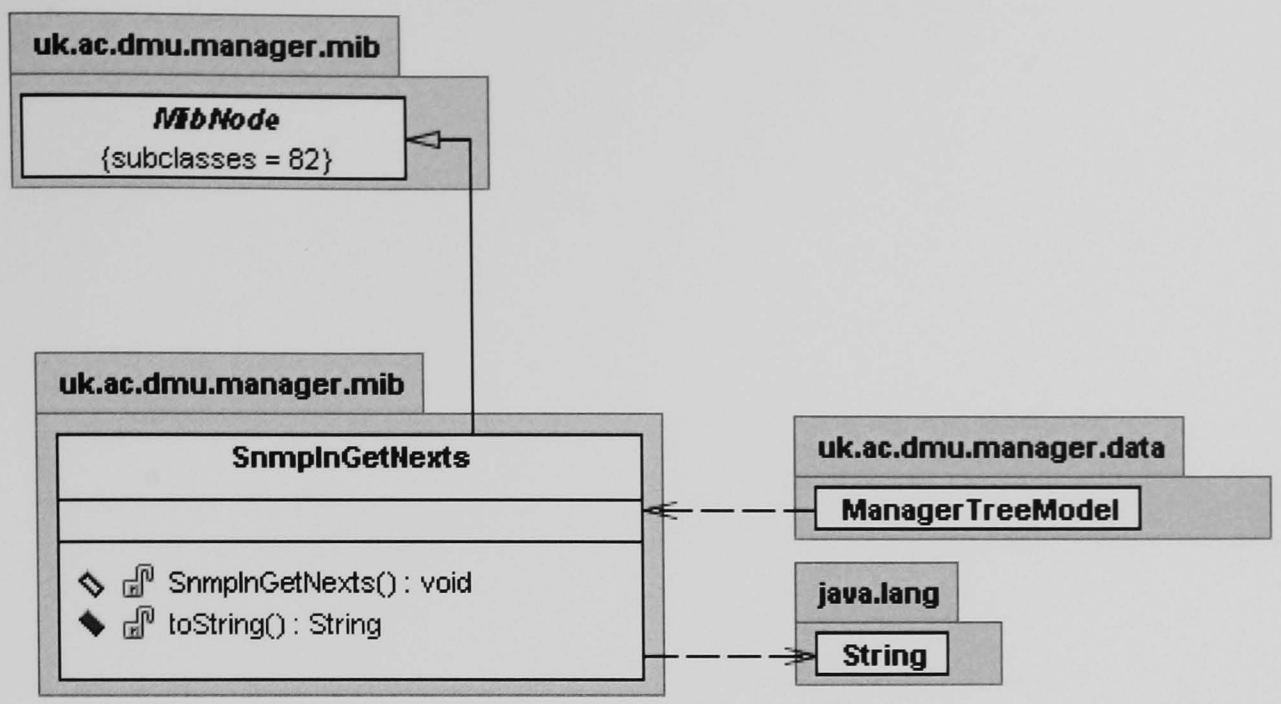
Class: SnmpInBadCommunityNames



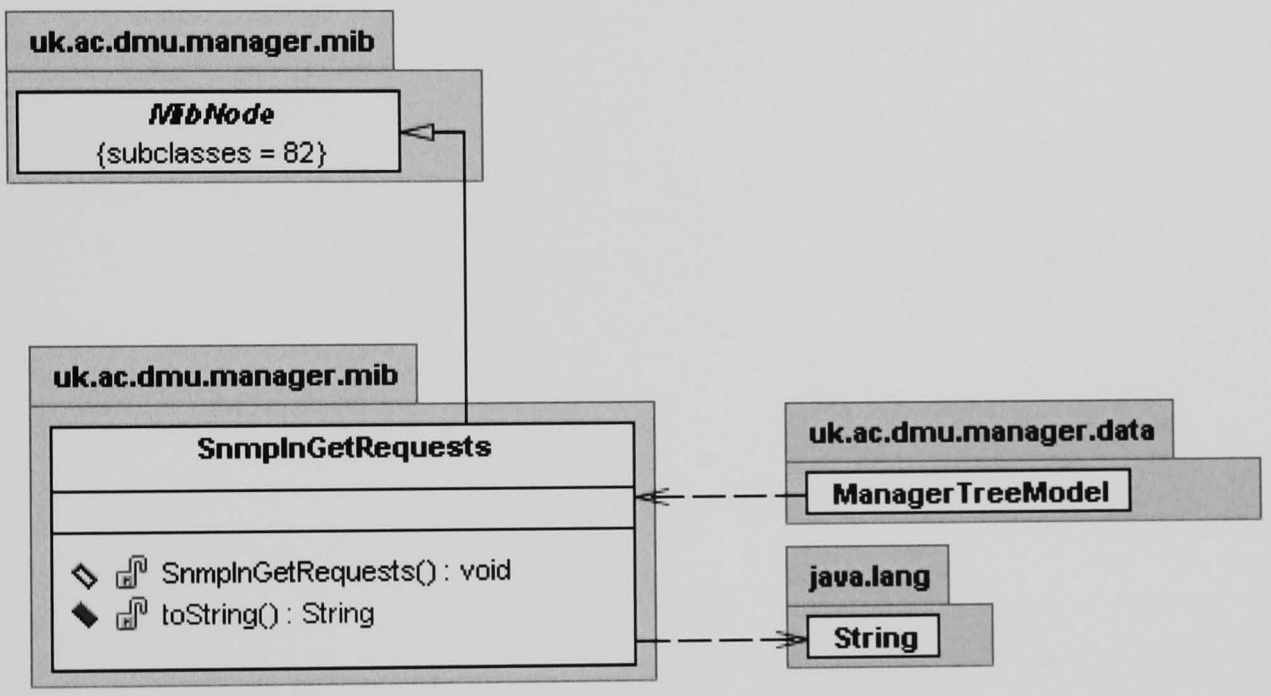
Class: SnmpInBadVersions



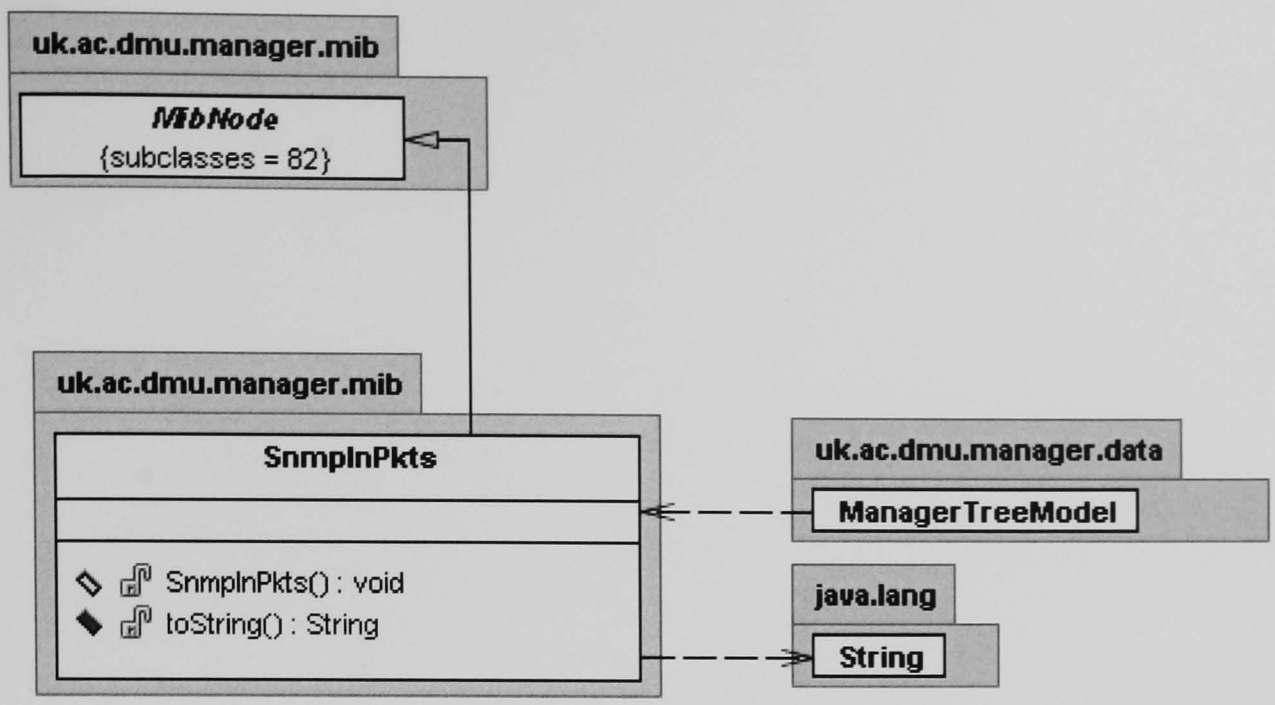
Class: SnmpInGetNexts



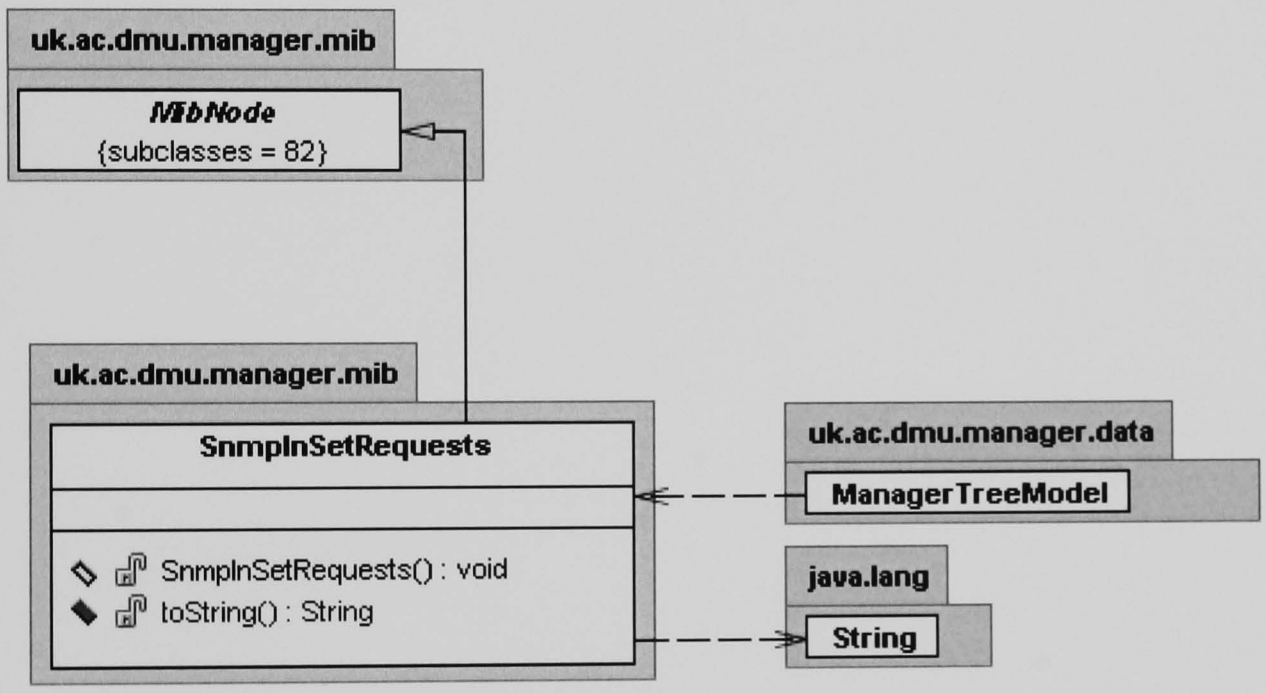
Class: SnmpInGetRequests



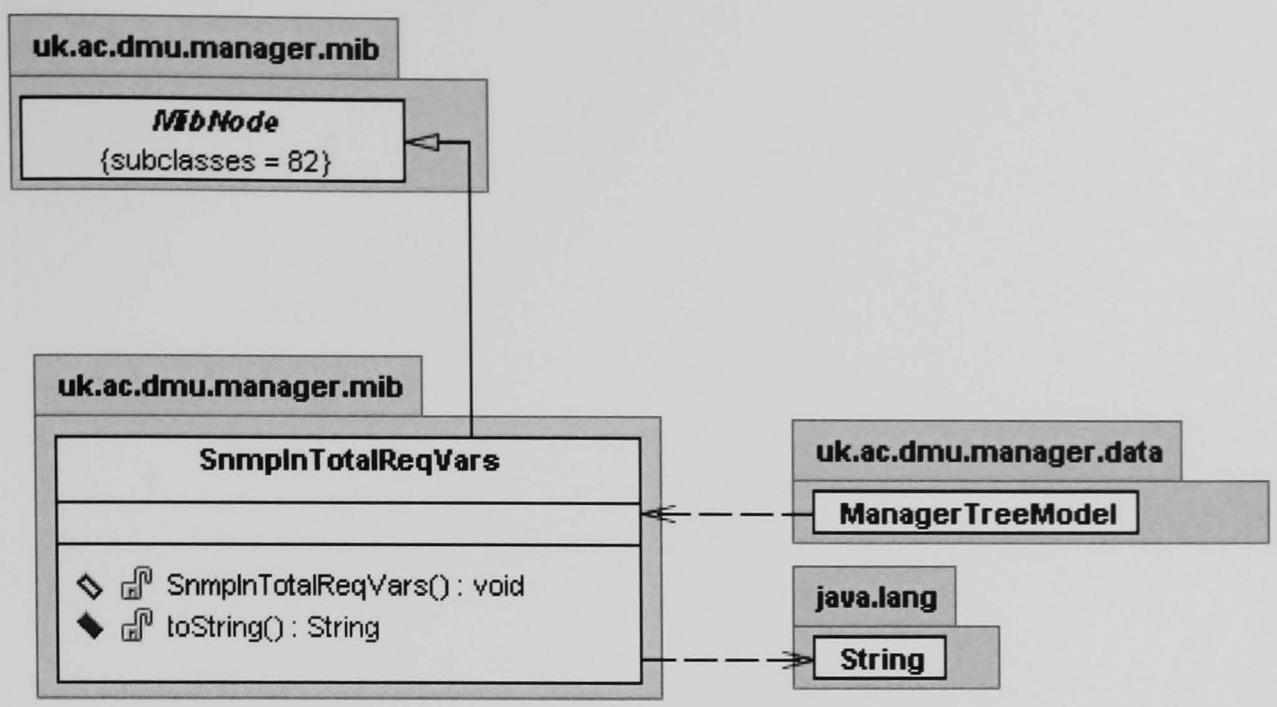
Class: SnmpInPkts



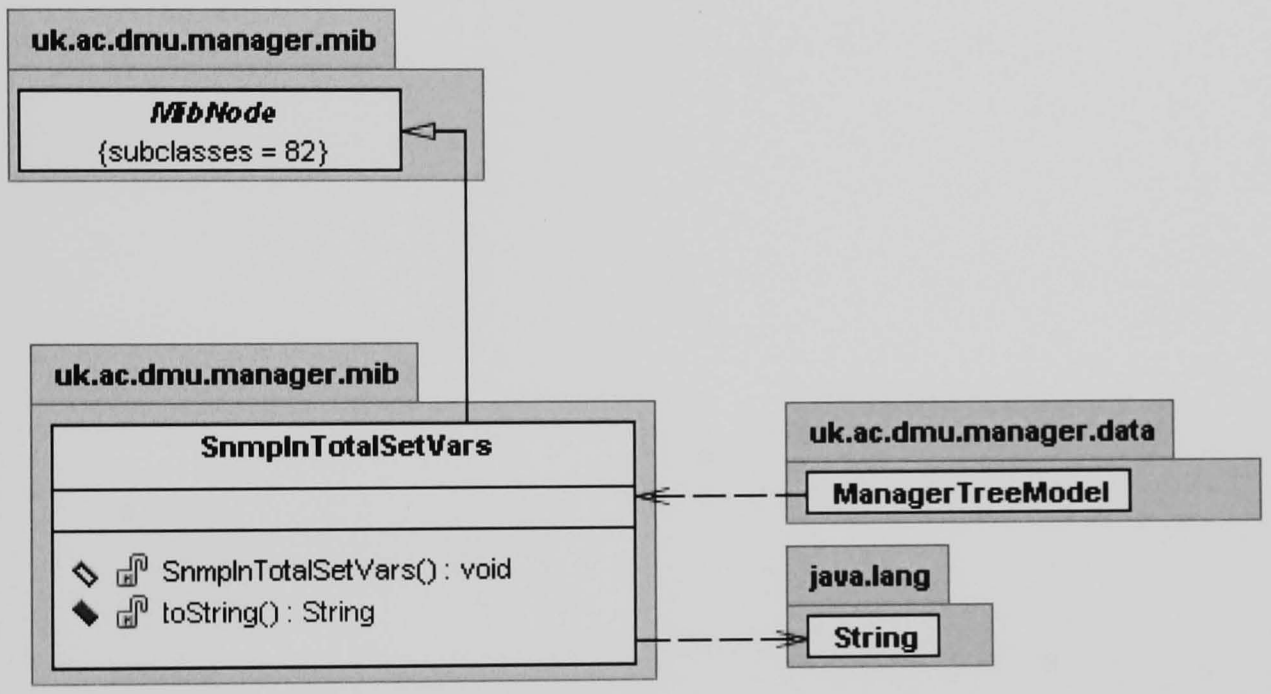
Class: SnmpInSetRequests



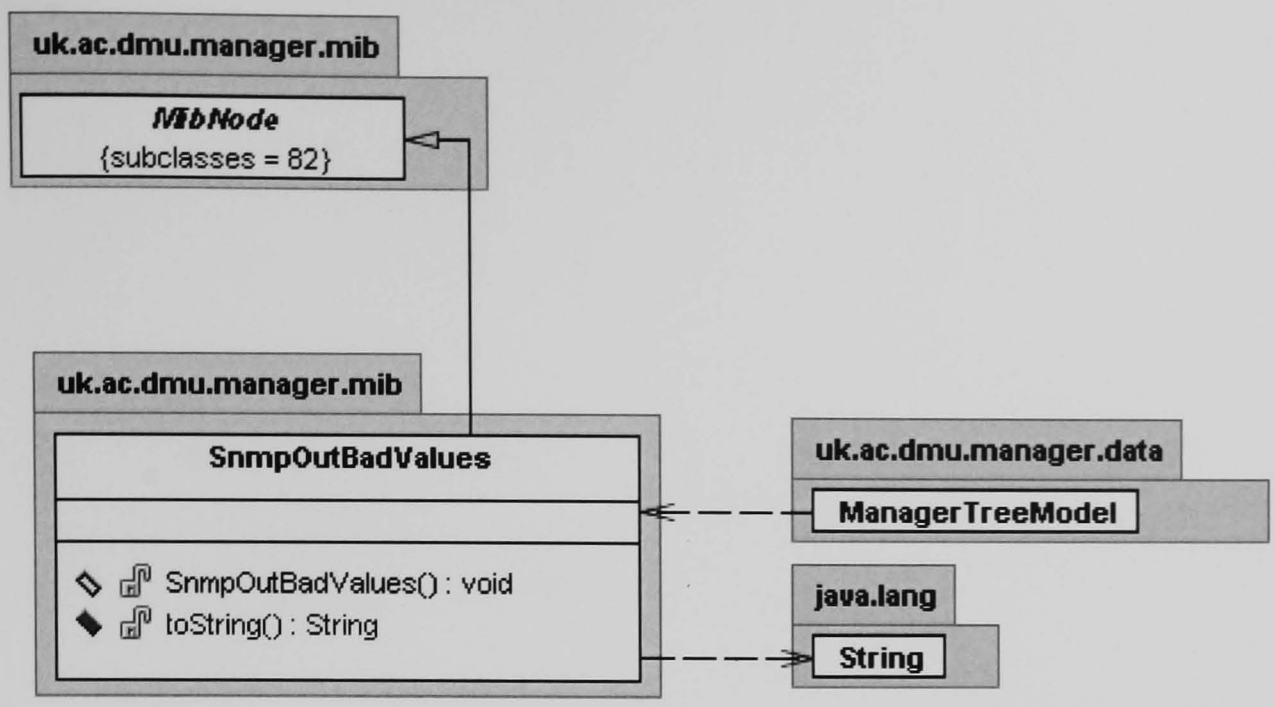
Class:SnmpInTotalReqVars



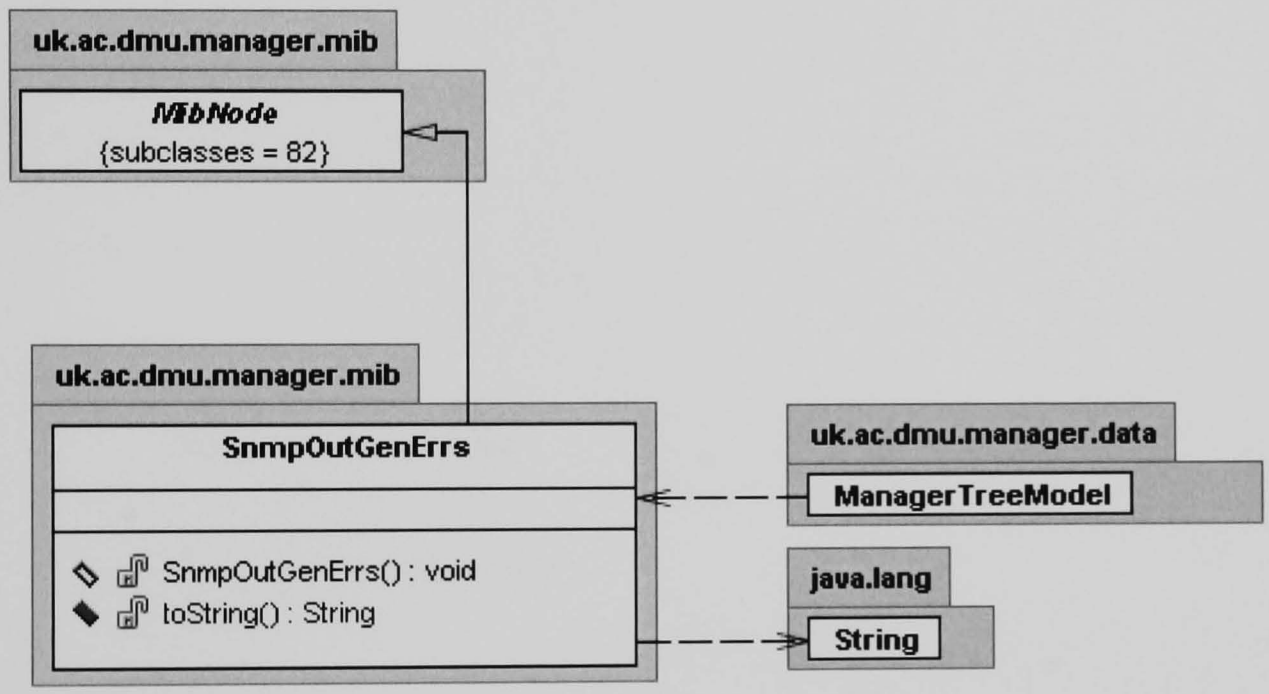
Class: SnmpInTotalSetVars



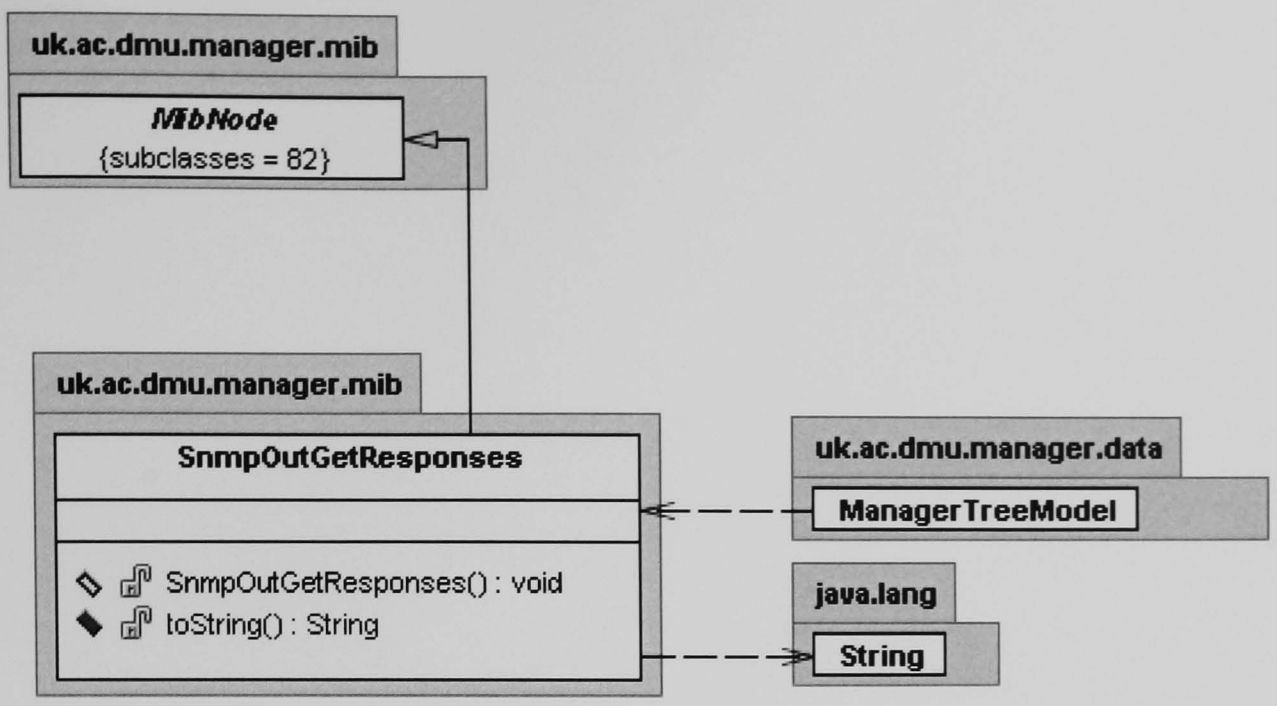
Class: SnmpOutBadValues



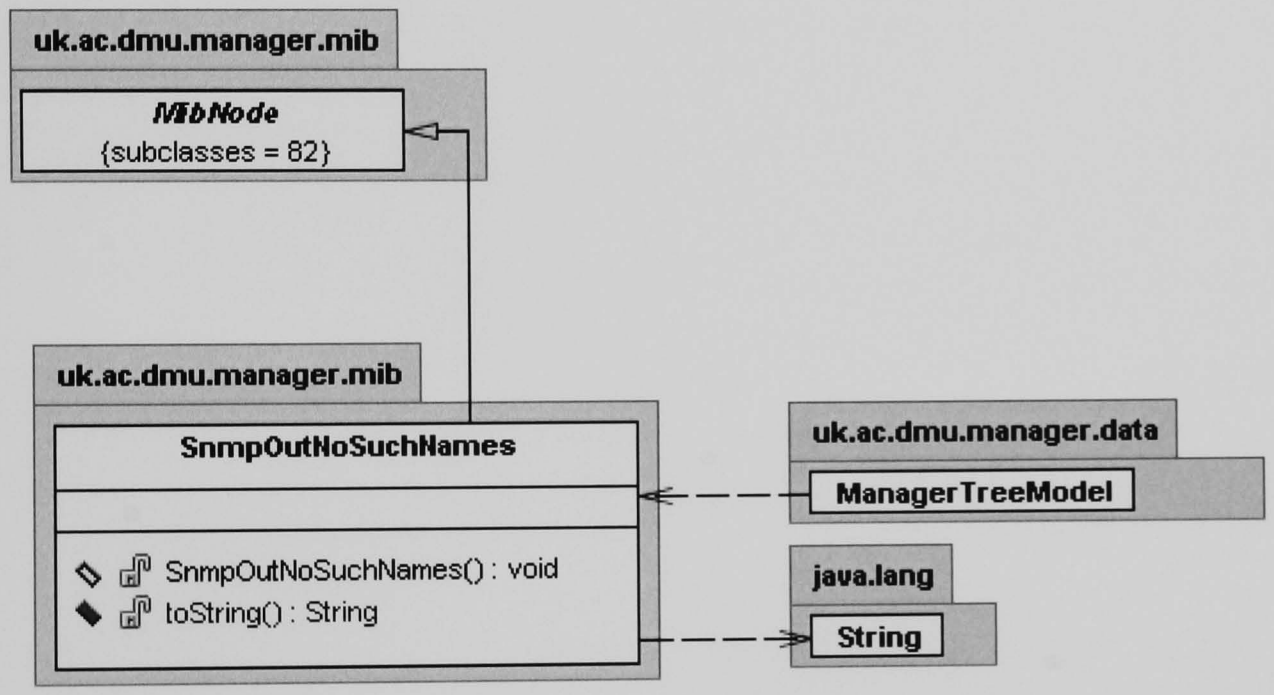
Class: SnmpOutGenErrs



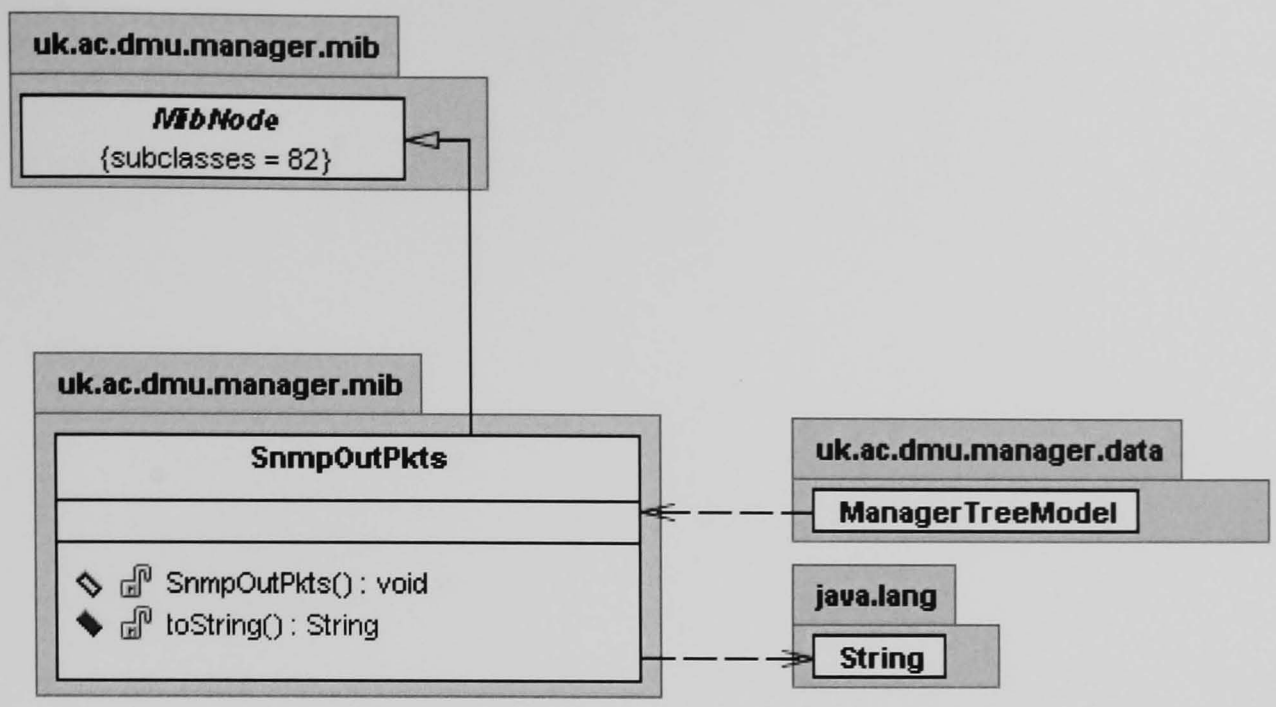
Class: SnmpOutGetResponses



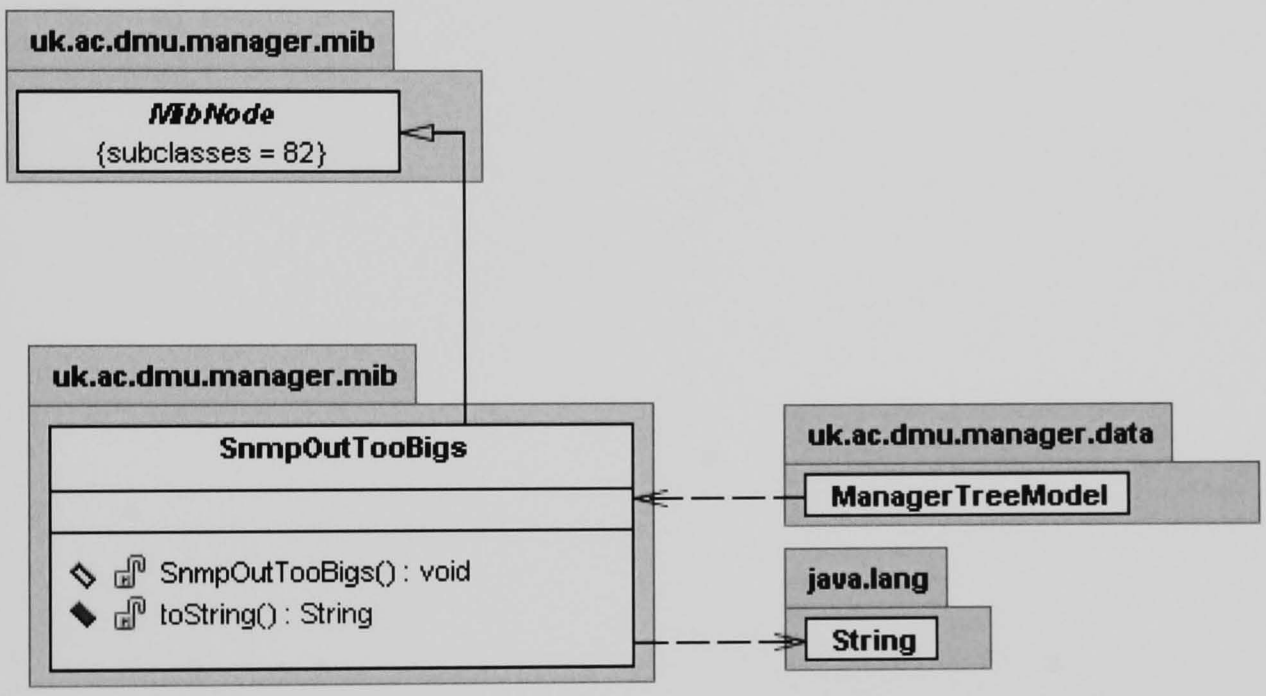
Class: SnmpOutNoSuchNames



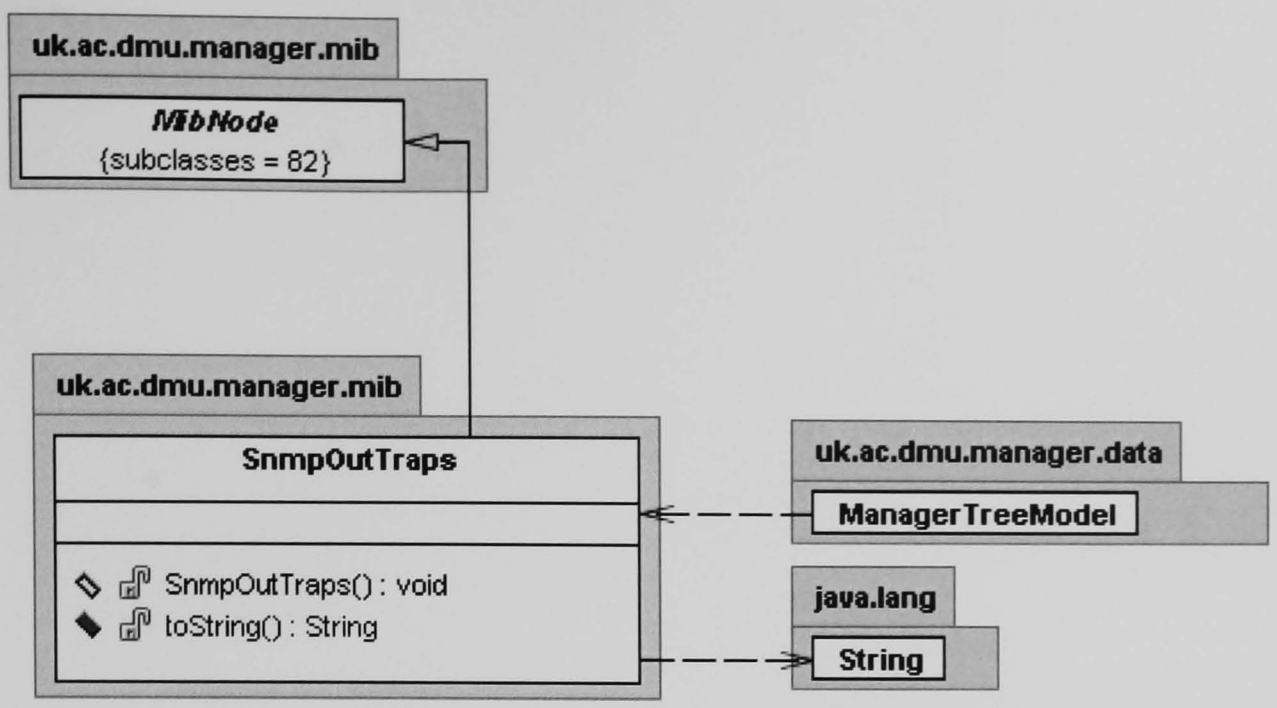
Class: SnmpOutPkts



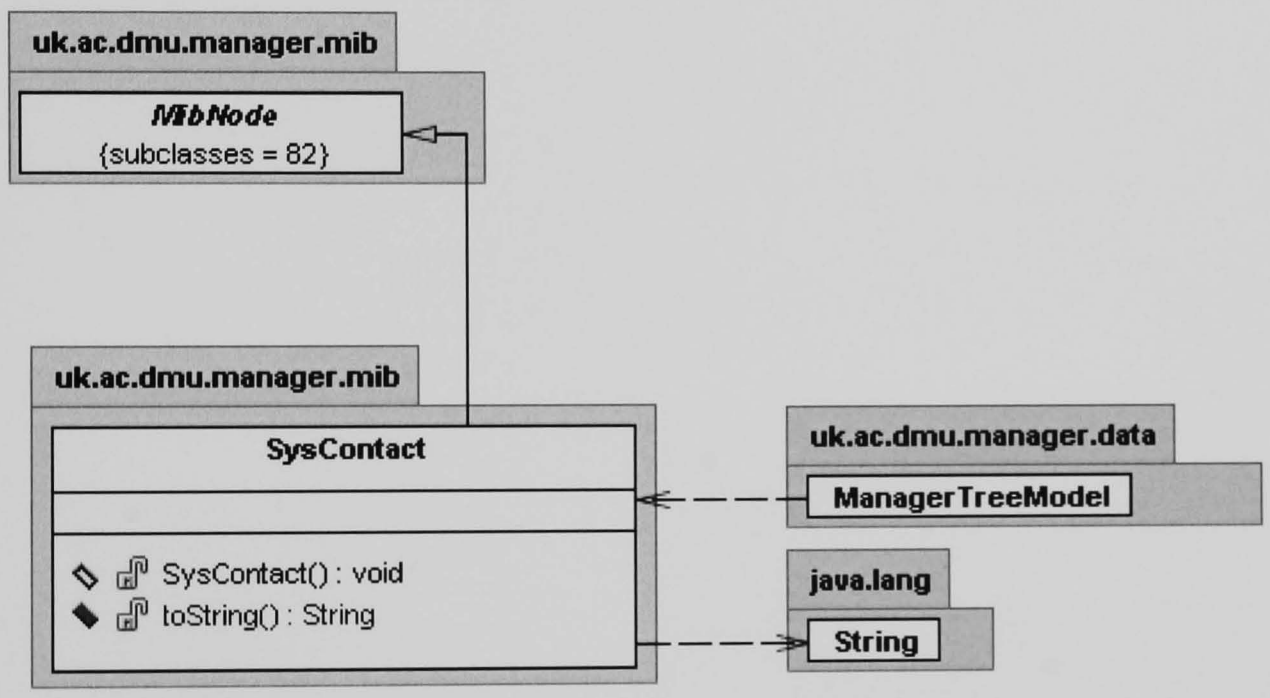
Class: SnmpOutTooBigs



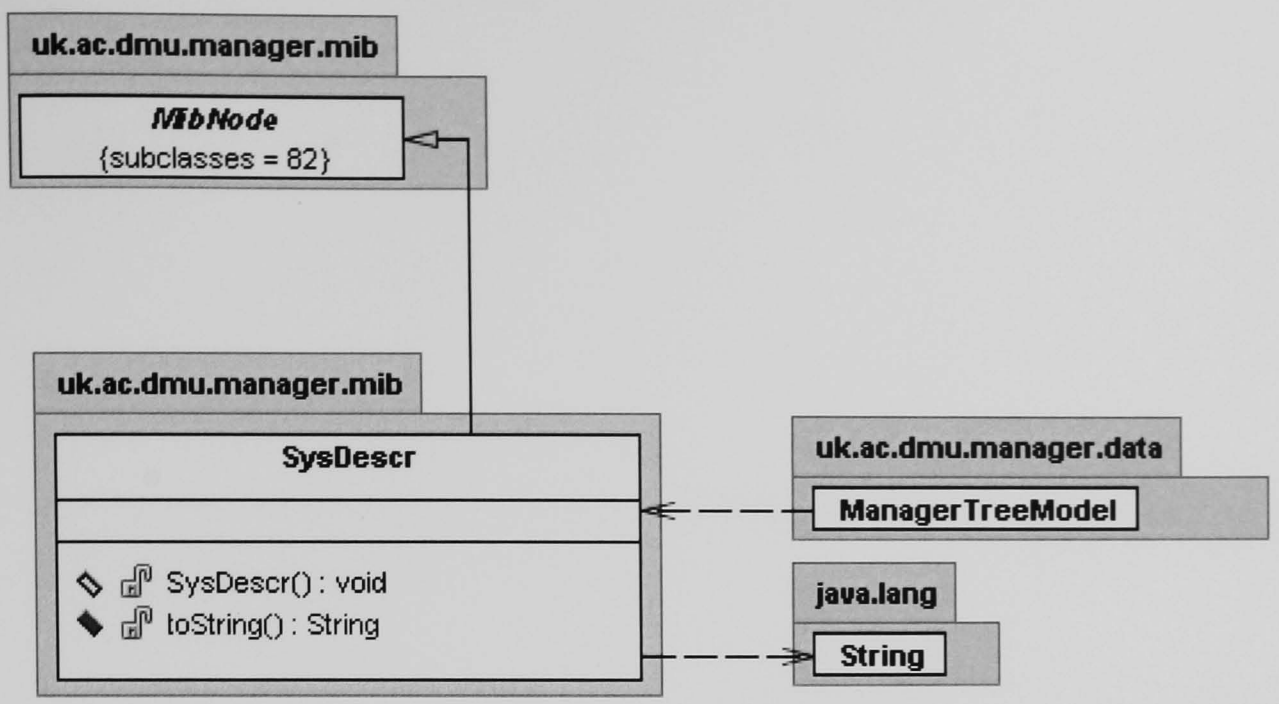
Class: SnmpOutTraps



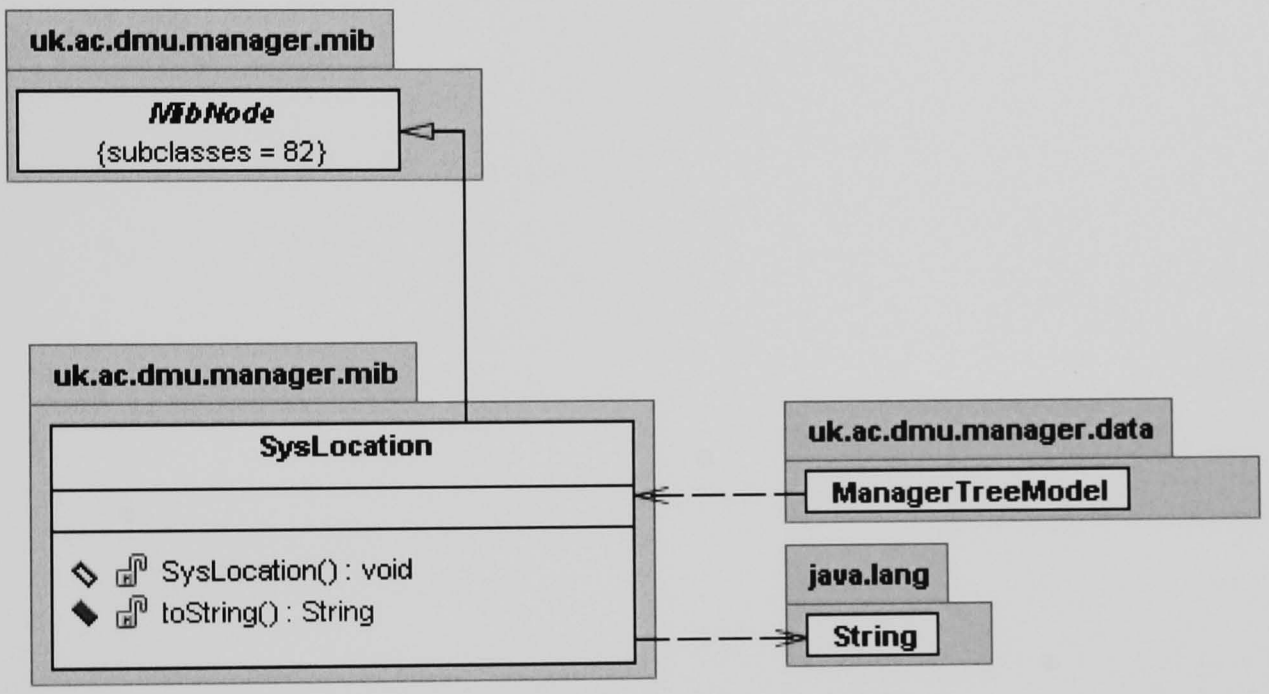
Class: SysContact



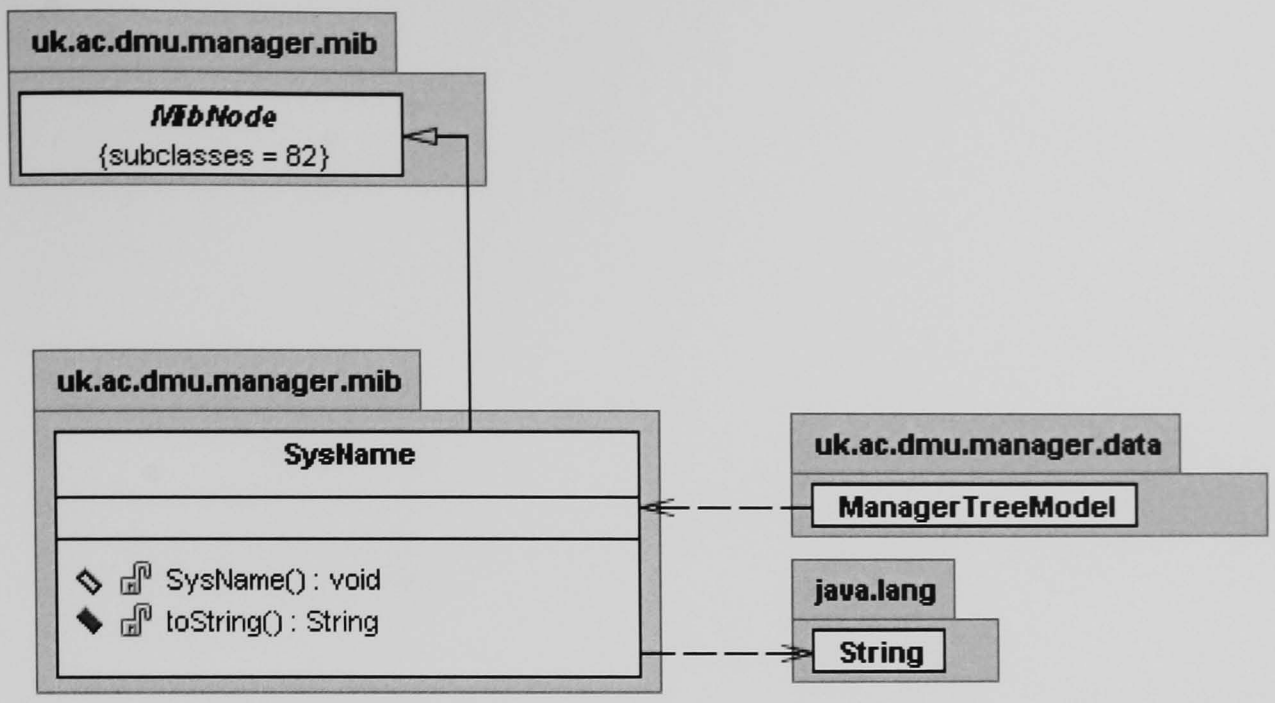
Class: SysDescr



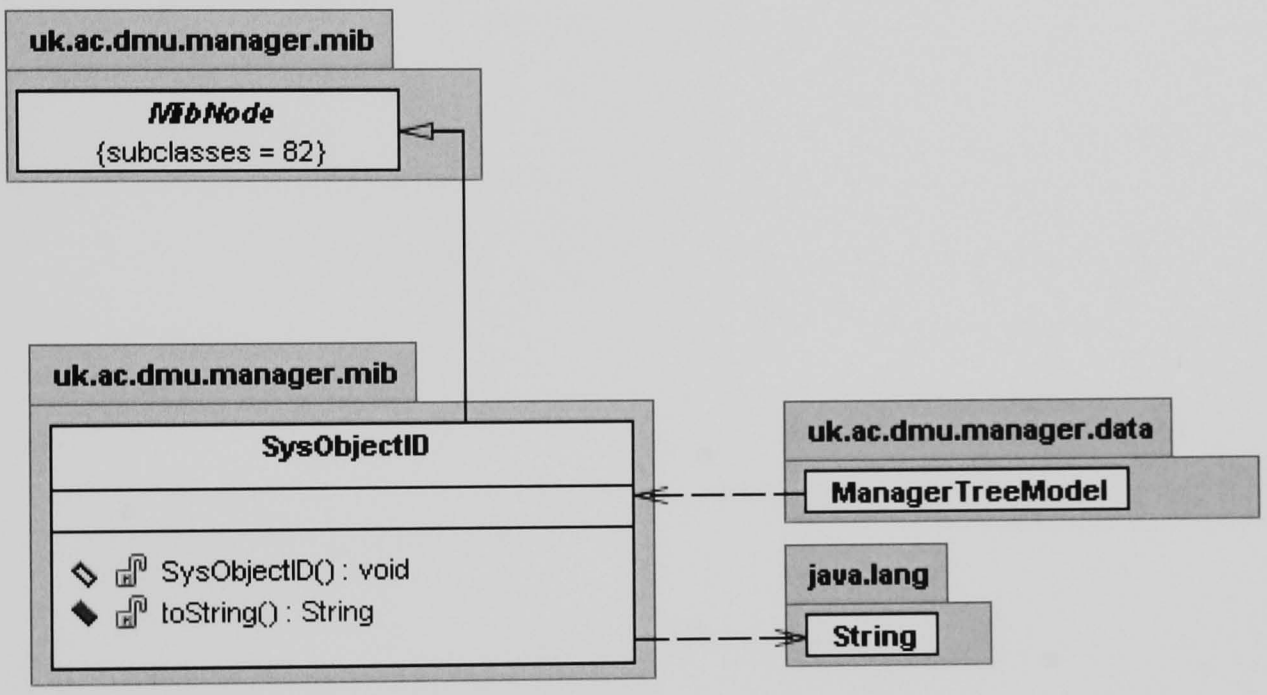
Class: SysLocation



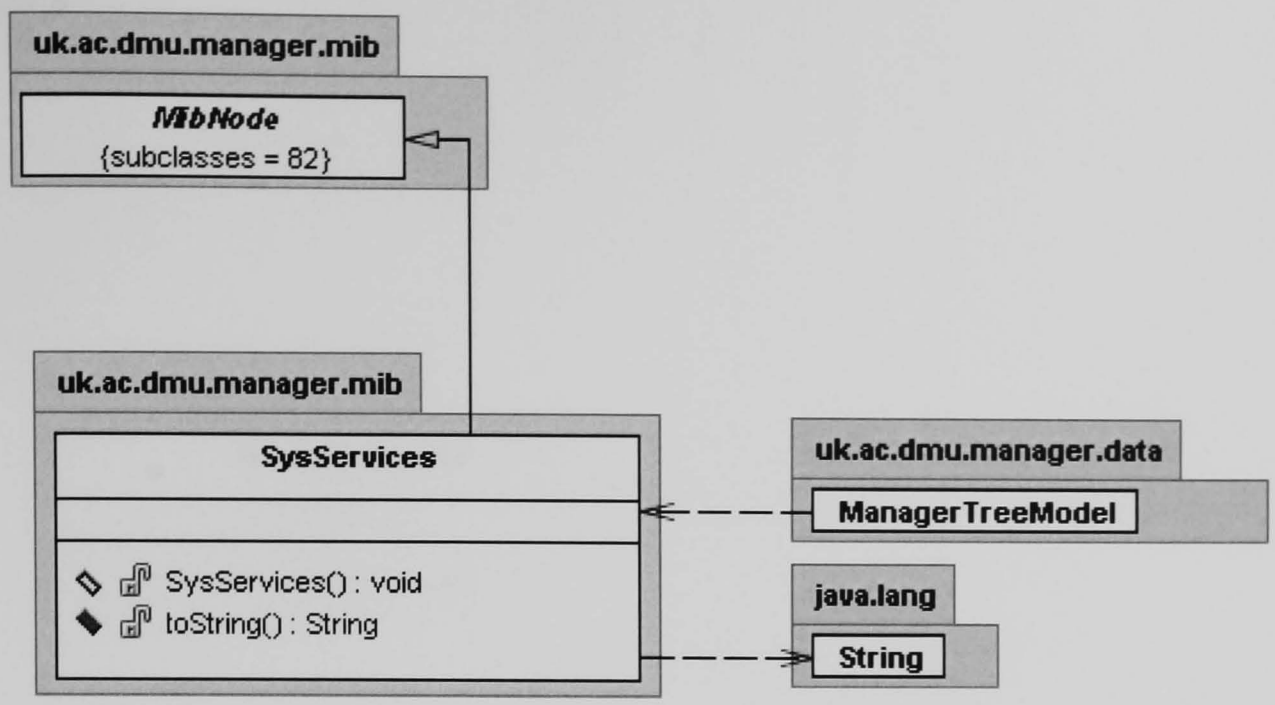
Class: SysName



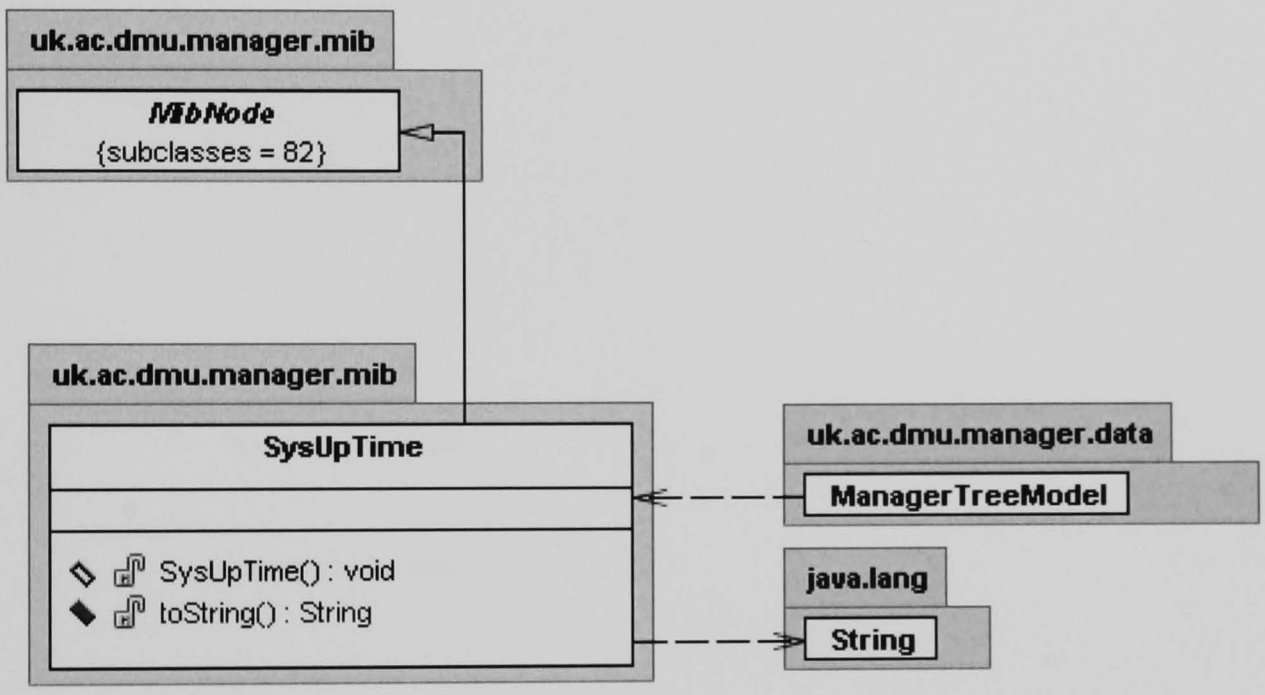
Class: SysObjectID



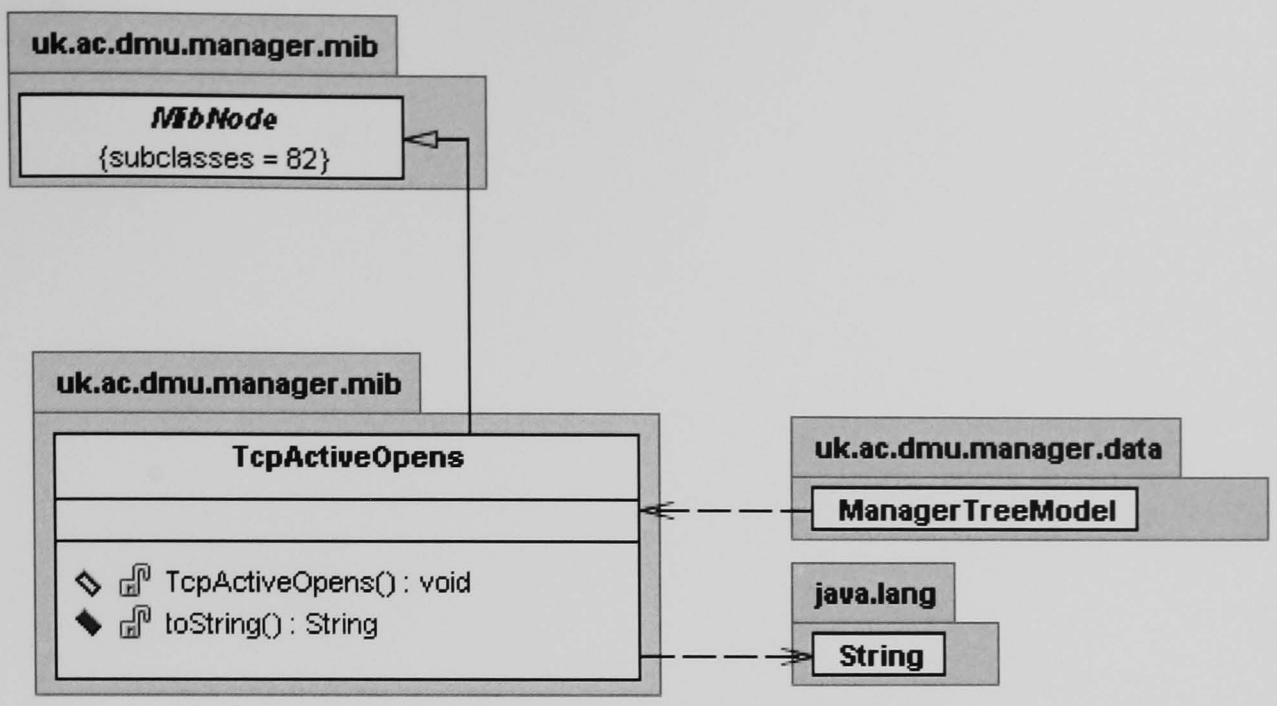
Class:SysServices



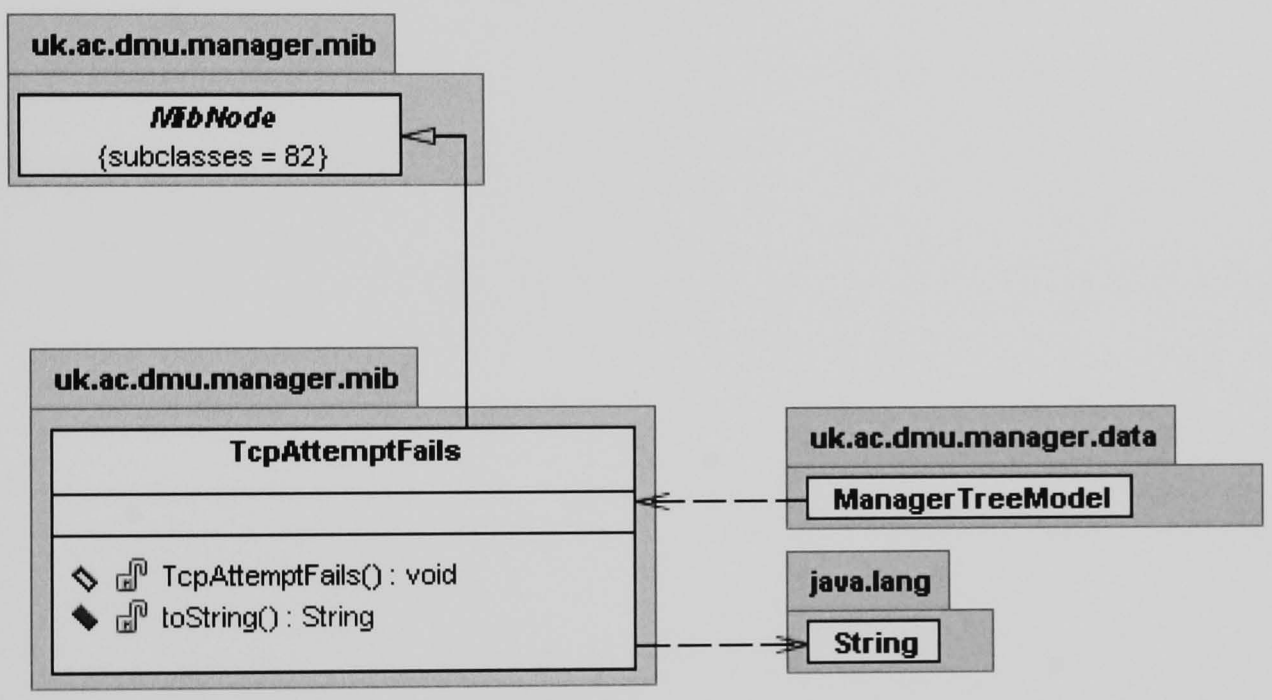
Class: SysUpTime



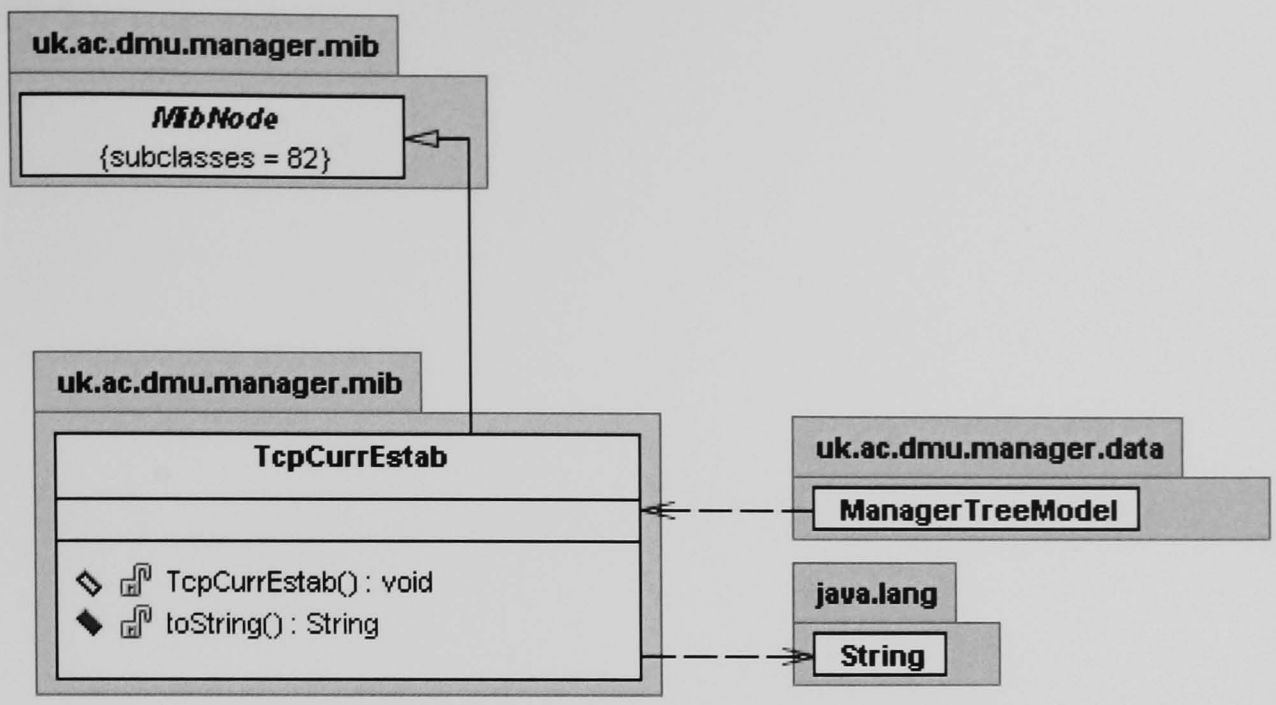
Class: TcpActiveOpens



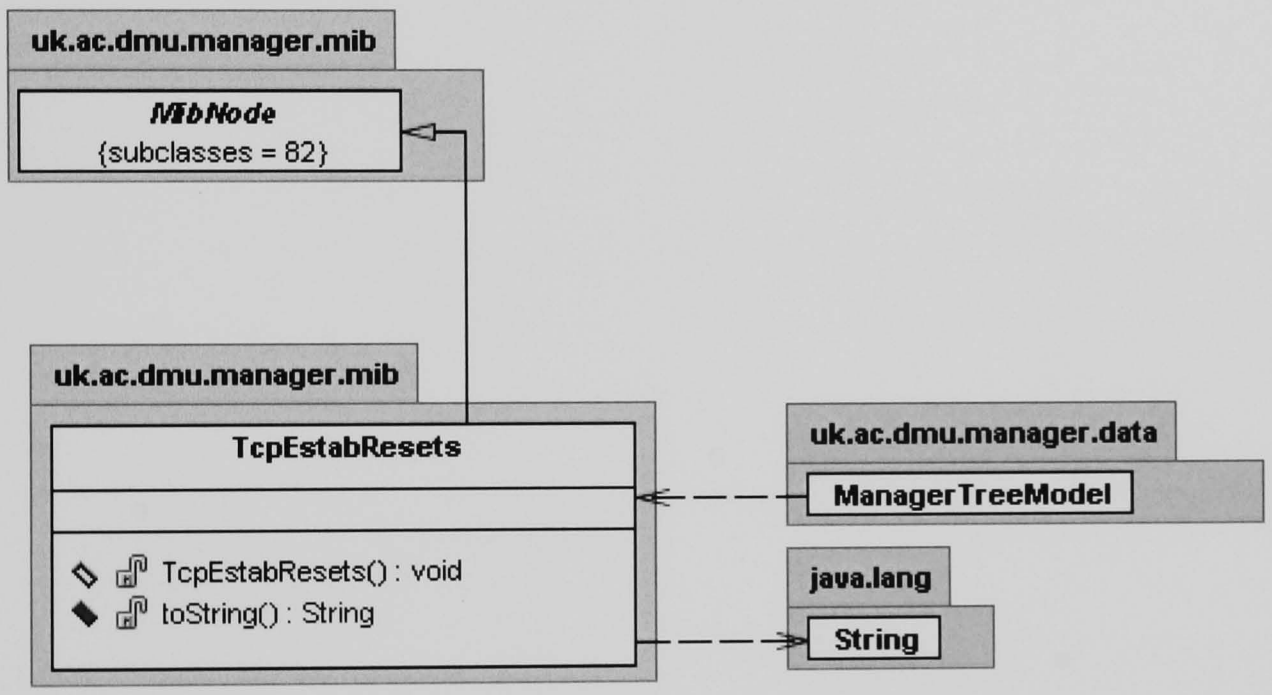
Class: TcpAttemptFails



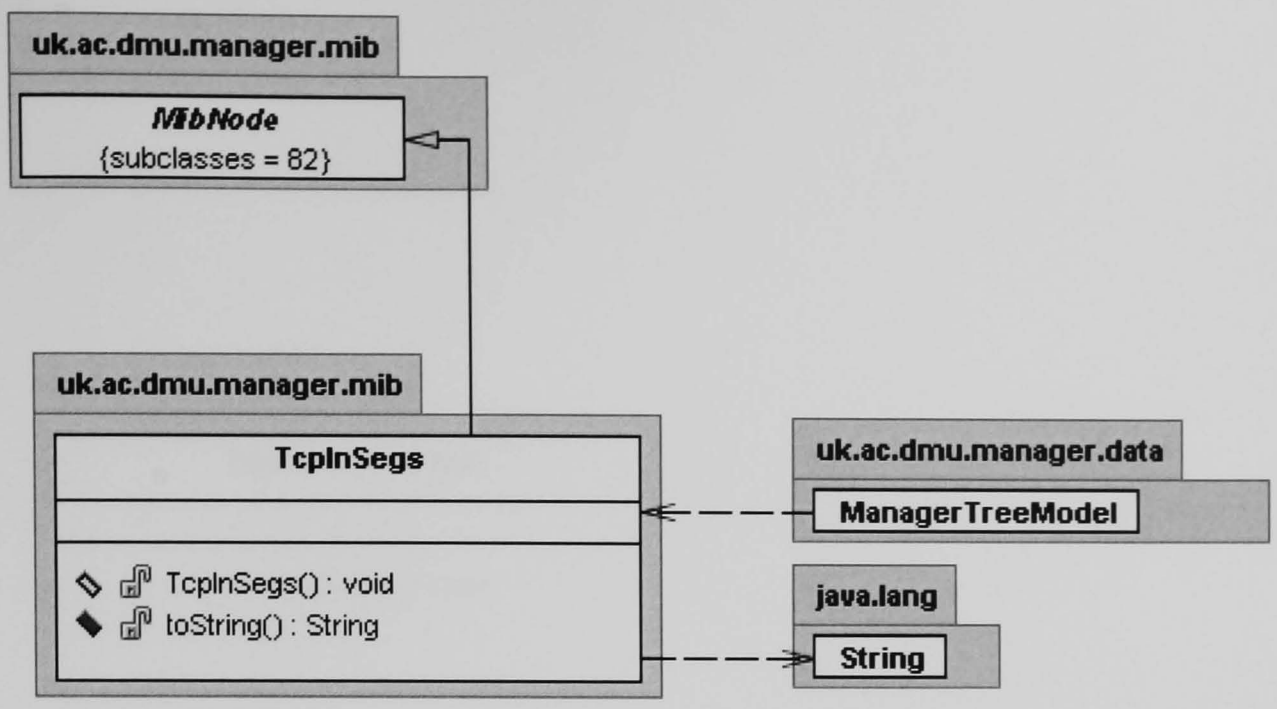
Class: TcpCurrEstab



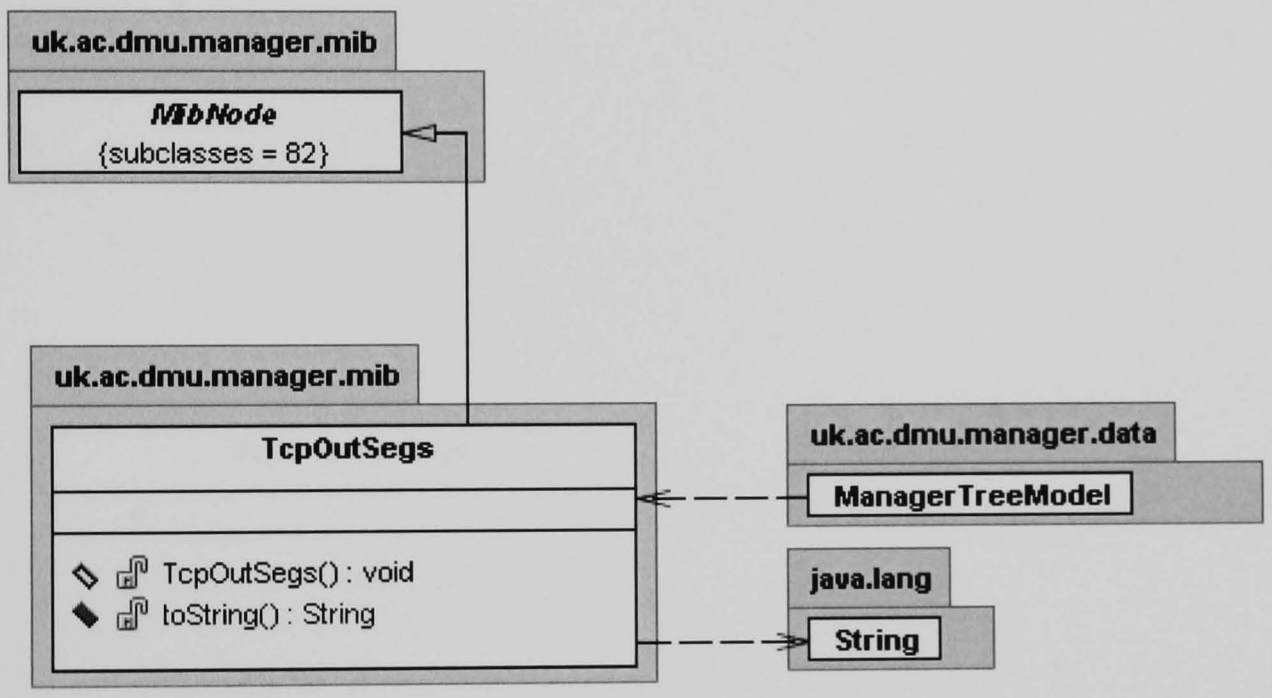
Class: TcpEstabResets



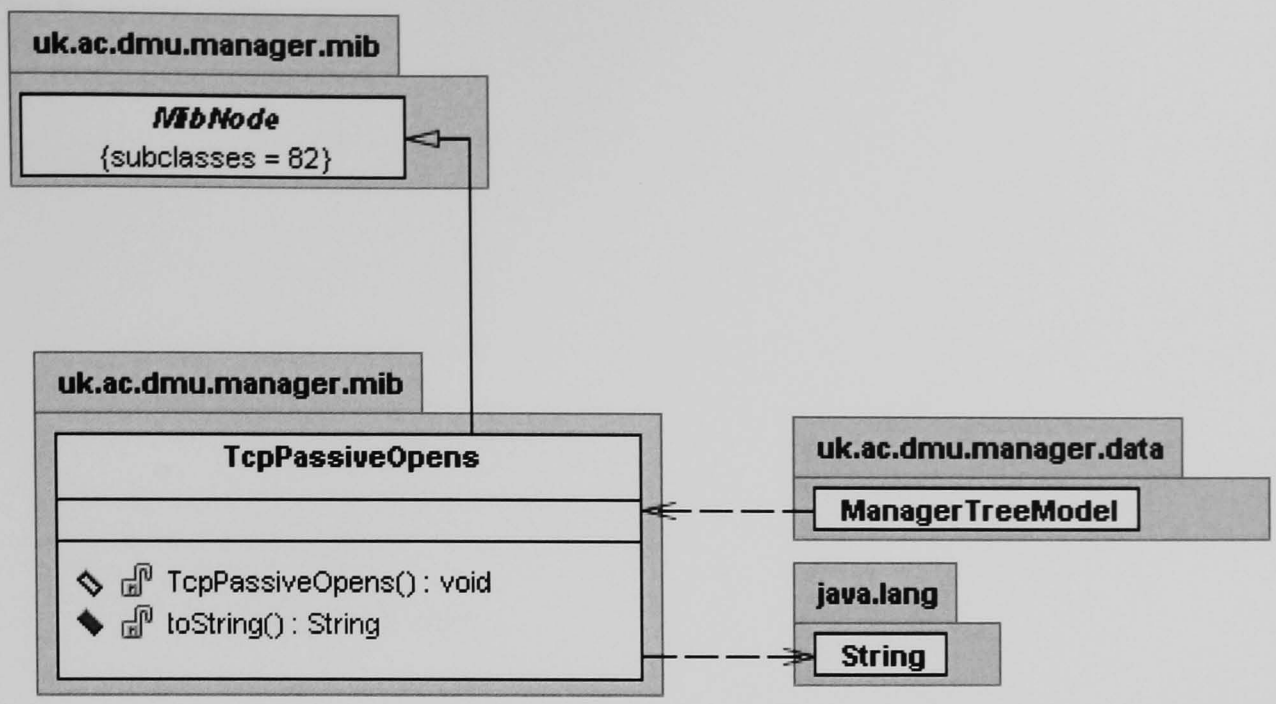
Class: TcpInSegs



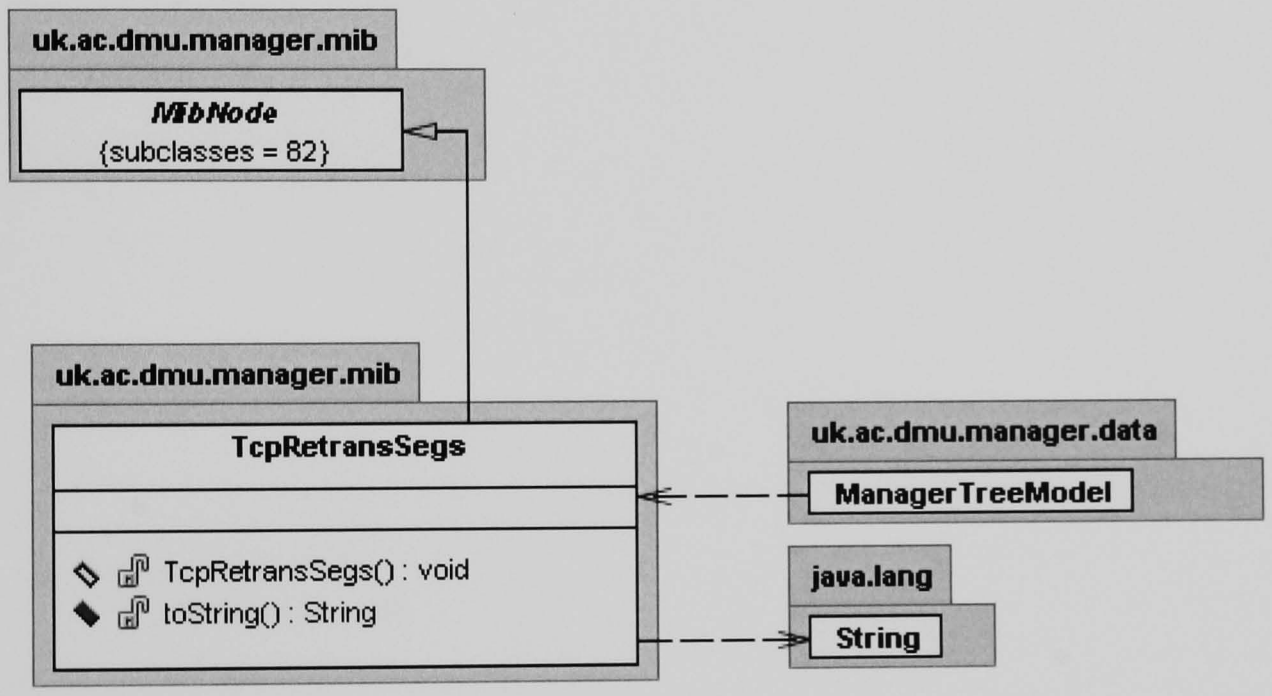
Class: TcpOutSegs



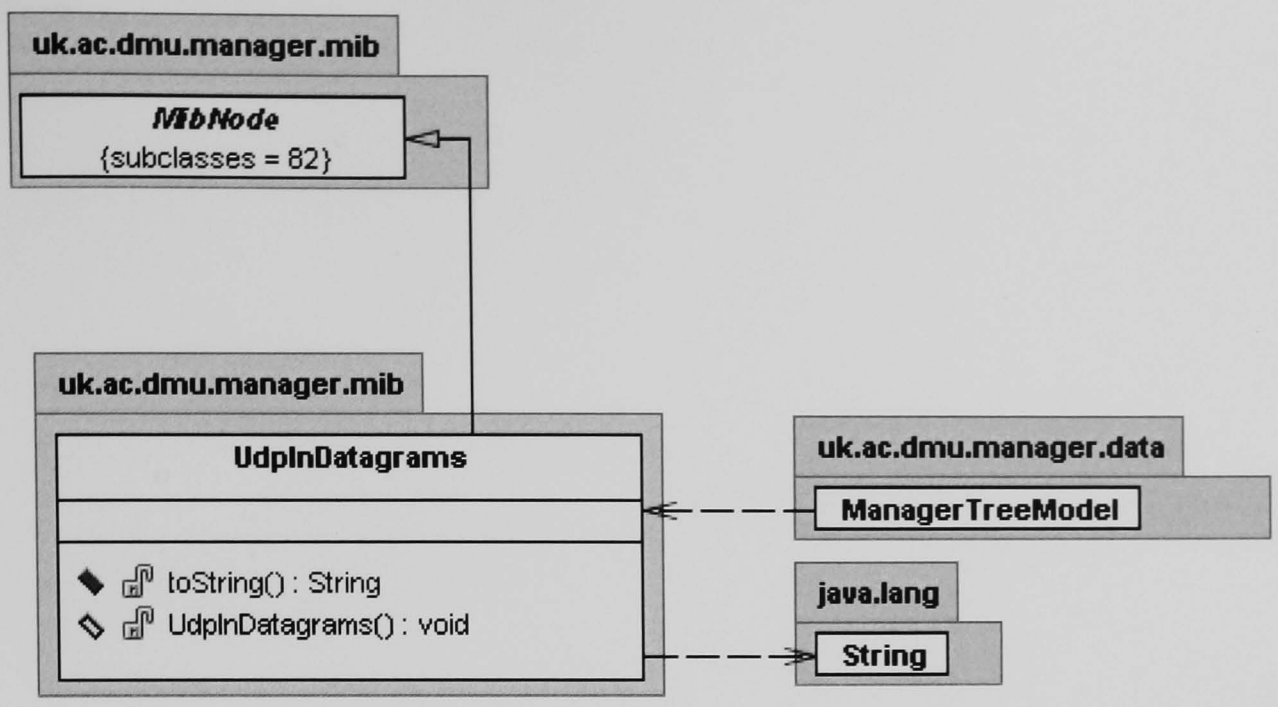
Class:TcpPassiveOpens



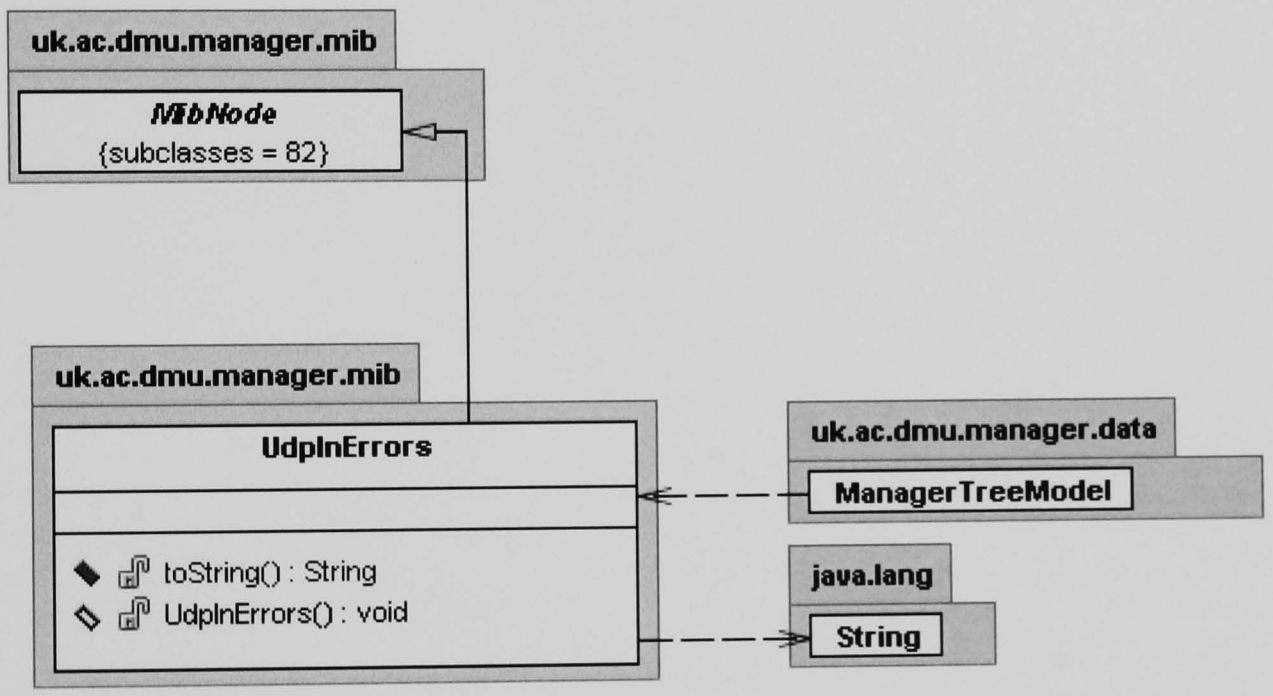
Class:TcpRetransSegs



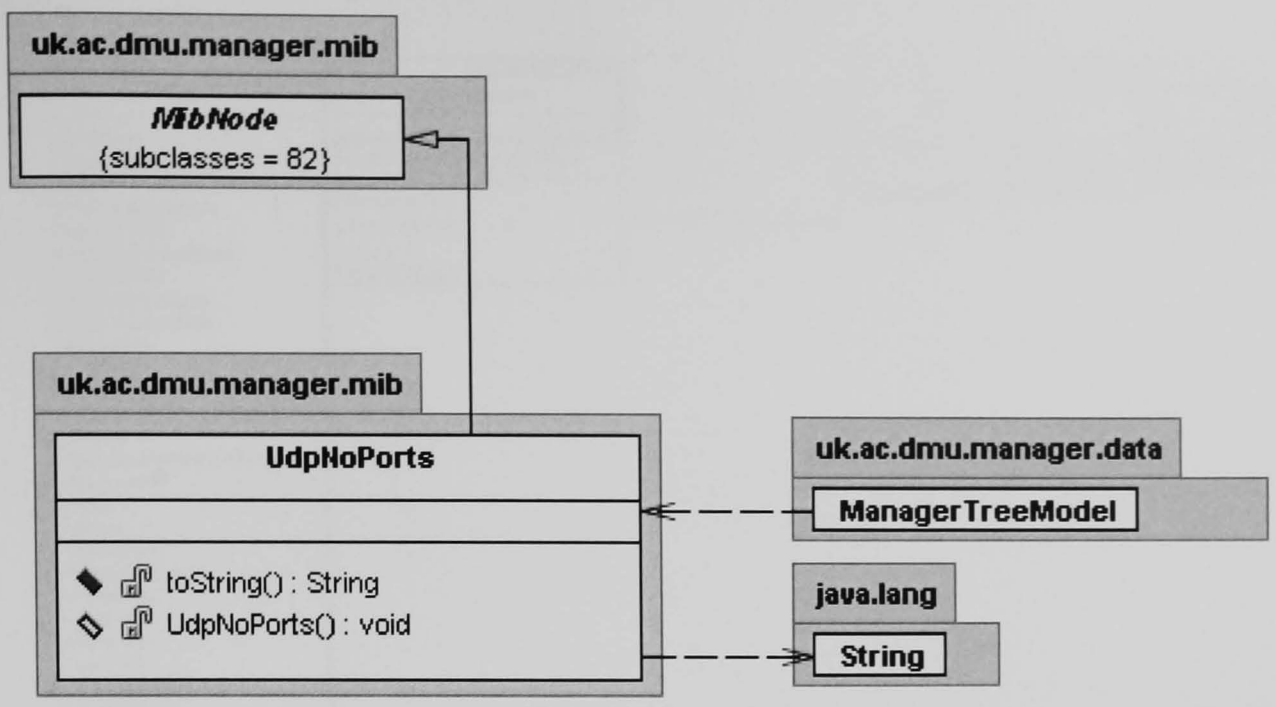
Class: UdpInDatagrams



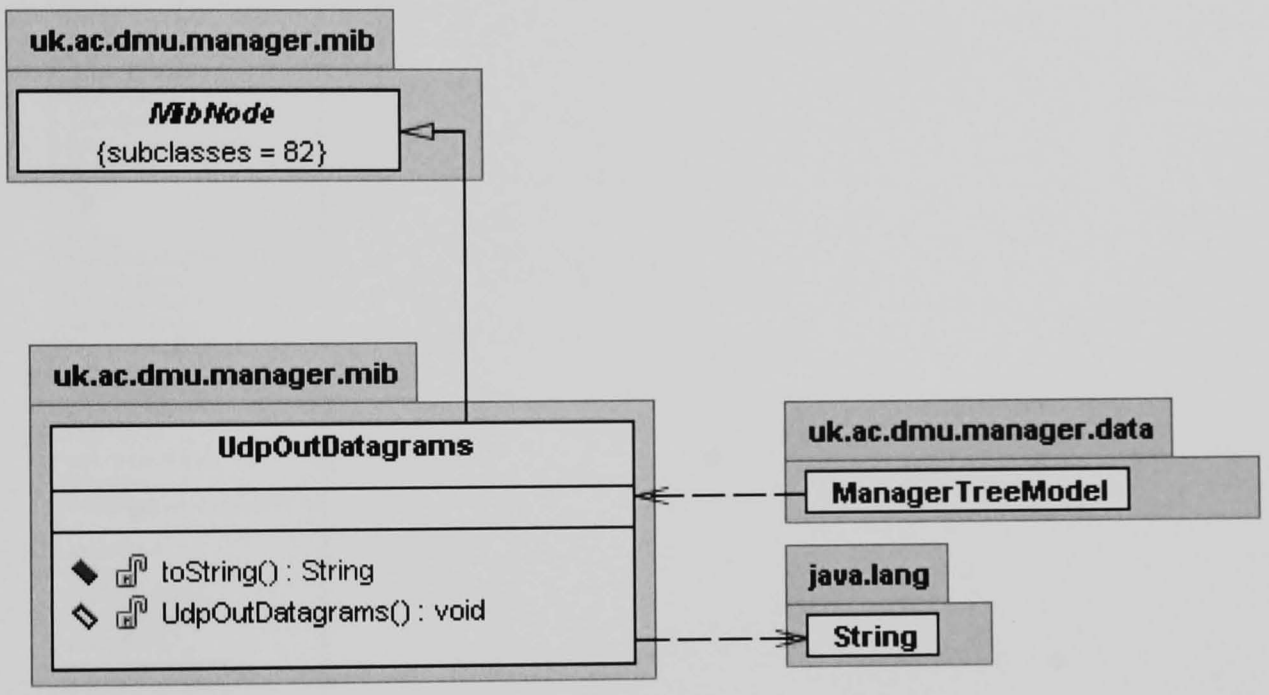
Class: UdpInErrors



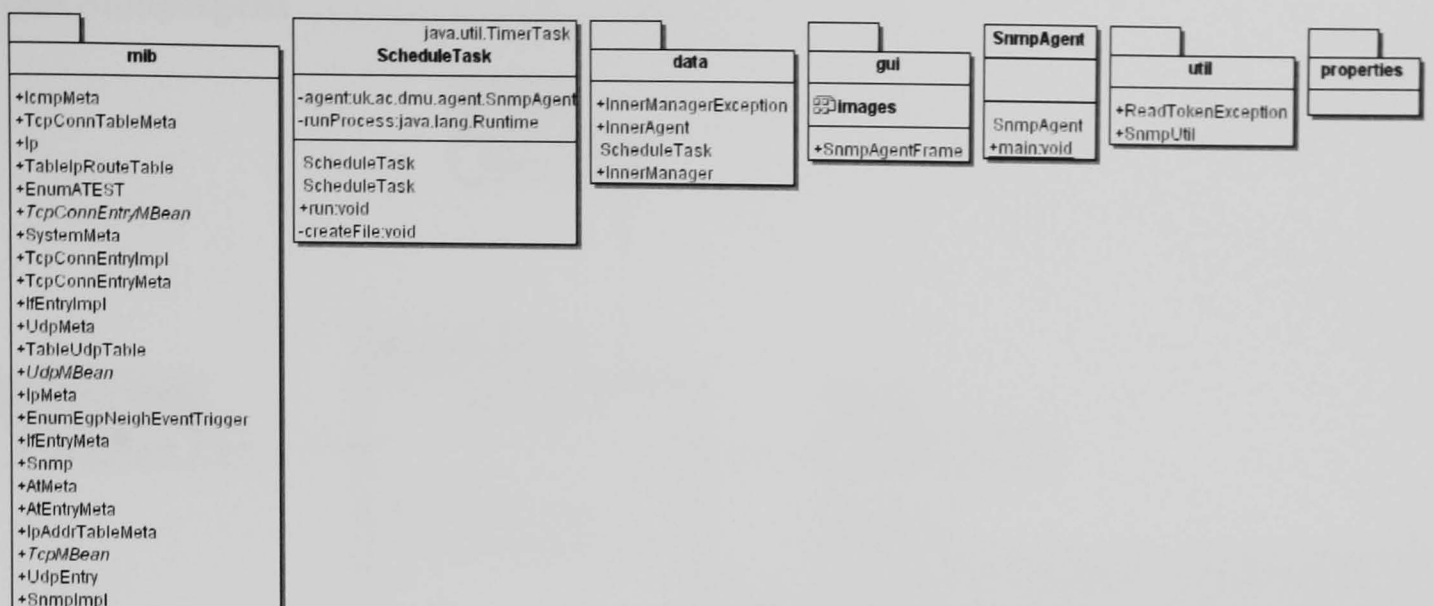
Class: UdpNoPorts



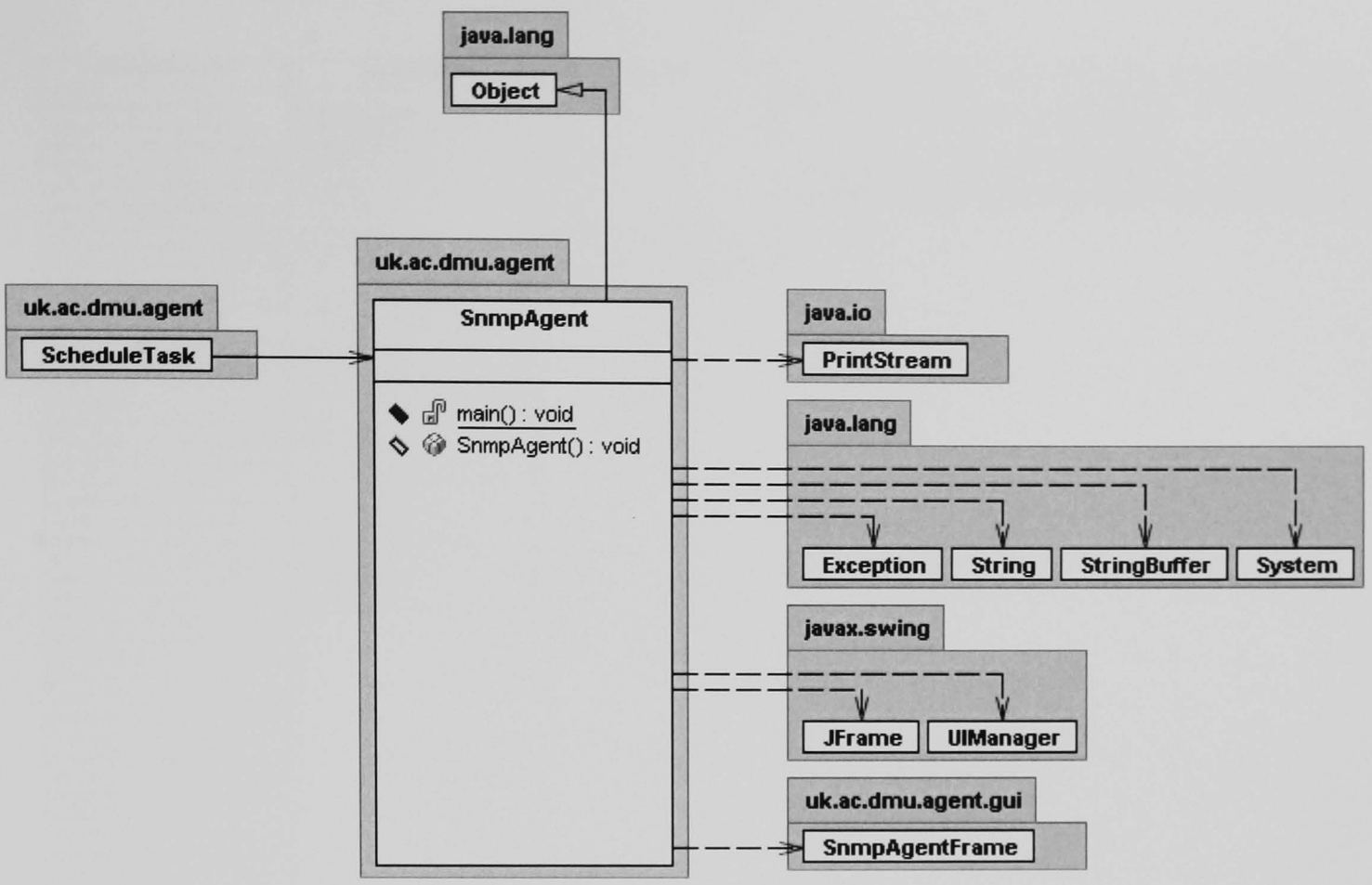
Class: UdpOutDatagrams



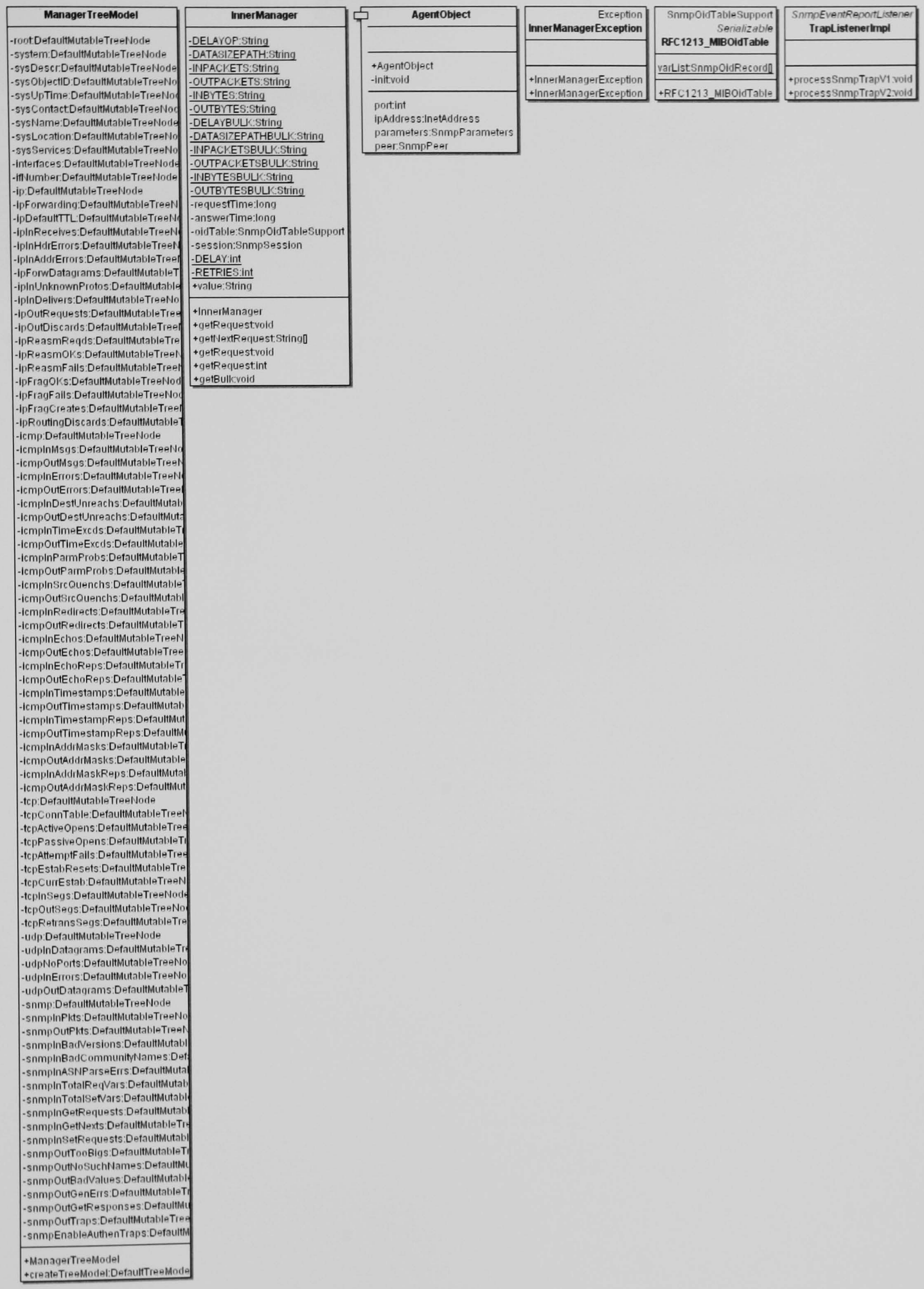
C.2 PACKAGE UK.AC.DMU.AGENT


mib
+IcmpMeta
+TcpConnTableMeta
+Ip
+TableIpRouteTable
+EnumATEST
+TcpConnEntryMBean
+SystemMeta
+TcpConnEntryImpl
+TcpConnEntryMeta
+IfEntryImpl
+UdpMeta
+TableUdpTable
+UdpMBean
+IpMeta
+EnumEgpNeighEventTrigger
+IfEntryMeta
+Snmp
+AtMeta
+AtEntryMeta
+IpAddrTableMeta
+TcpMBean
+UdpEntry
+SnmpImpl
+IpMBean
+UdpEntryMBean
+SnmpMBean
+AtTableMeta
+InterfacesMeta
+TableIpNetToMediaTable
+IpNetToMediaEntryMBean
+SnmpMeta
+EnumIpNetToMediaType
+IfEntry
+Interfaces
+EnumTcpRtoAlgorithm
+TcpImpl
+UdpImpl
+Udp
+IpAddrEntryMeta
+TableAtTable
+IcmpMBean
+IpNetToMediaEntryMeta
+SystemImpl
+AtEntry
+Tcp
+Icmp
+EnumTcpConnState
+UdpTableMeta
+TableIpAddrTable
+IpRouteEntryMeta
+RFC1213 MIBOidTable
+EnumEgpNeighMode
+TableTcpConnTable
+RFC1213 MIB
+TableIfTable
+IpAddrEntryMBean
+IfTableMeta
+EnumSnmpEnableAuthenTraps
+At
+EnumIfType
+IpImpl
+EnumIpRouteType
+IpNetToMediaEntry
+InterfacesImpl
+EnumEgpNeighState
+EnumIfAdminStatus
+TcpConnEntry
+TcpMeta
+System
+UdpEntryMeta
+IpRouteTableMeta
+SystemMBean
+IpNetToMediaTableMeta
+AtMBean
+IpRouteEntryMBean
+InterfacesMBean
+EnumIfOperStatus
+IcmpImpl
+EnumIpRouteProto
+IpAddrEntry
+IpRouteEntry
+AtEntryMBean
+IfEntryMBean
java.util.TimerTask
ScheduleTask
-agent:uk.ac.dmu.agent.SnmpAgent
-runProcess:java.lang.Runtime
ScheduleTask
ScheduleTask
+run:void
-createFile:void
data
+InnerManagerException
+InnerAgent
ScheduleTask
+InnerManager
gui
Images
+SnmpAgentFrame
SnmpAgent
SnmpAgent
+main:void
util
+ReadTokenException
+SnmpUtil
properties

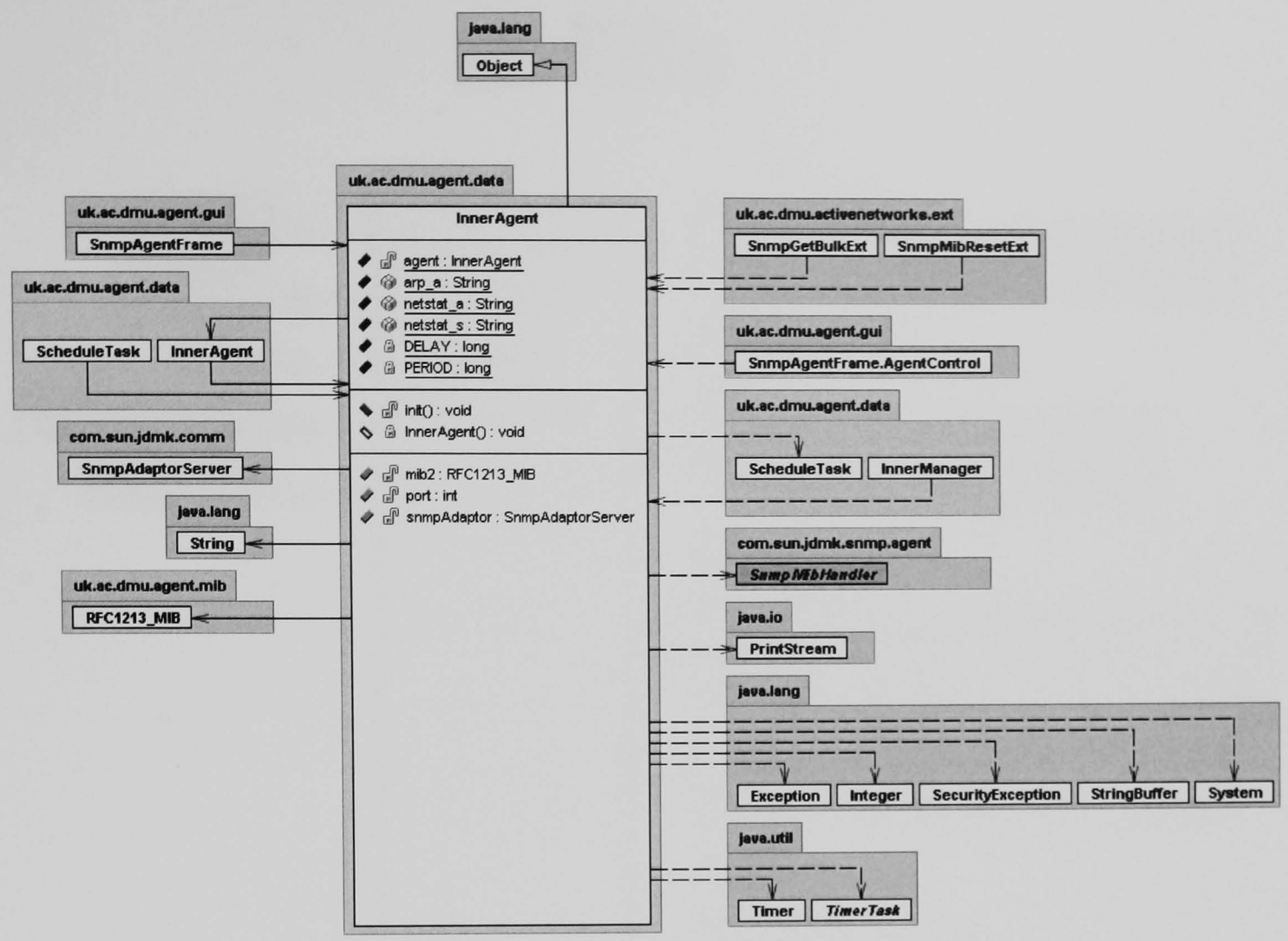
Class SnmpAgent



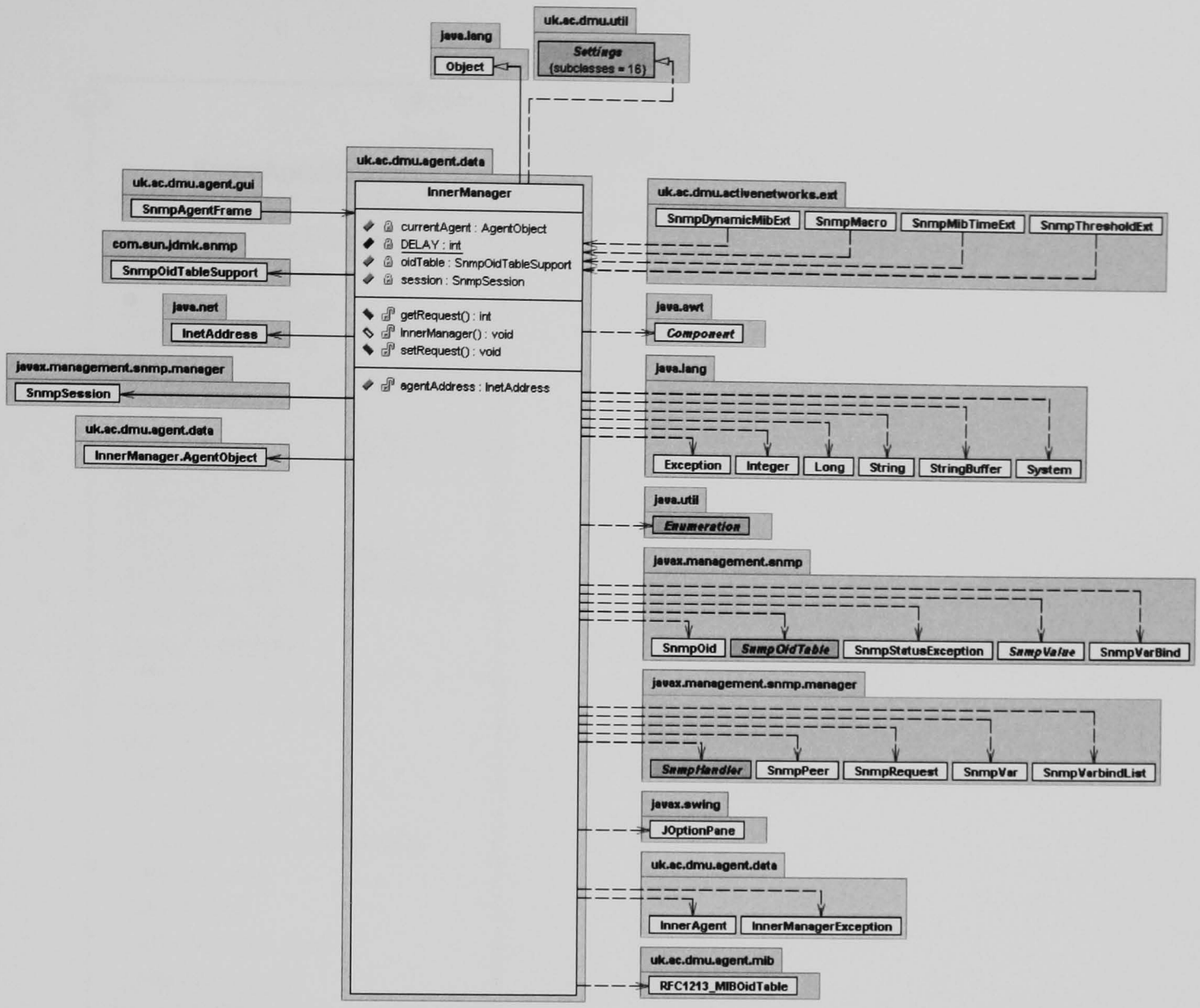
Package uk.ac.dmu.agent.data



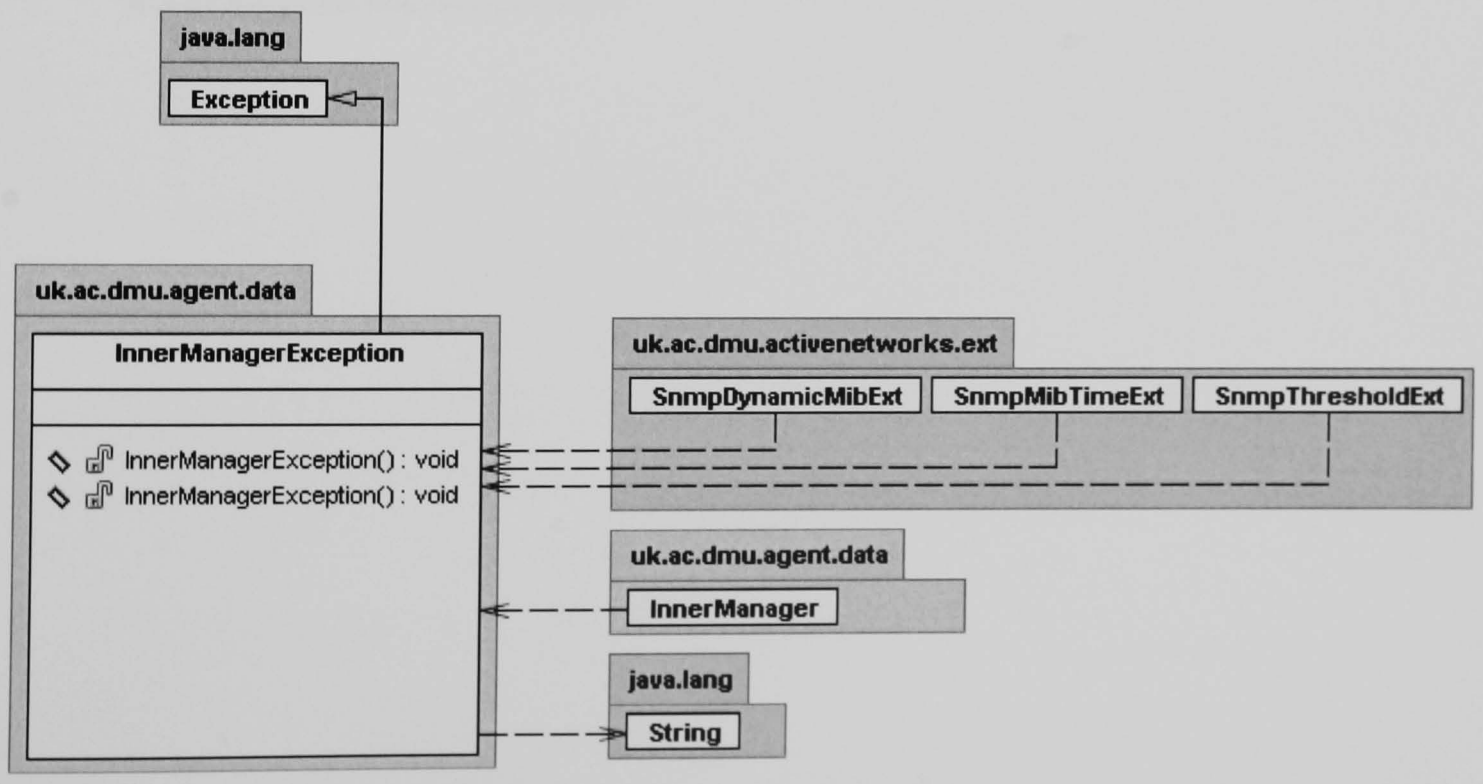
Class: InnerAgent



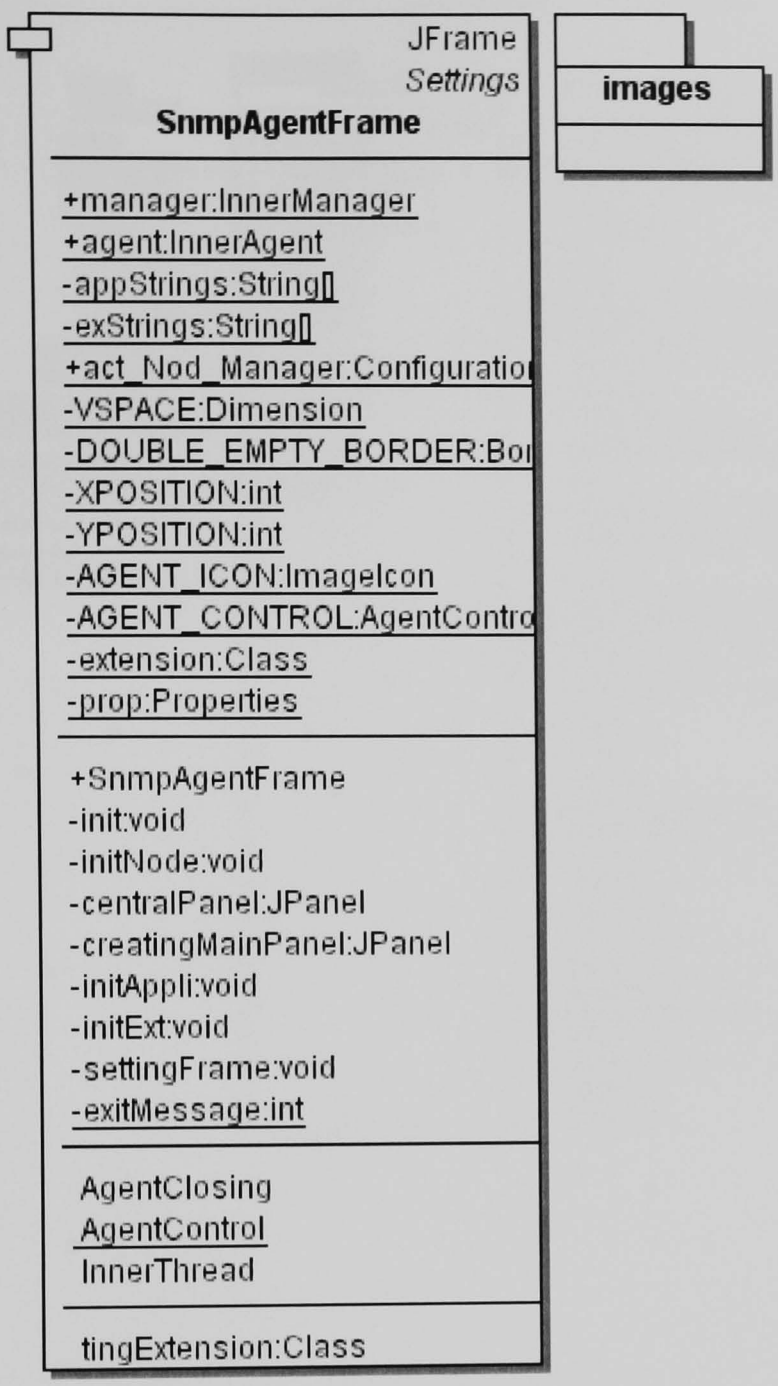
Class: InnerManager



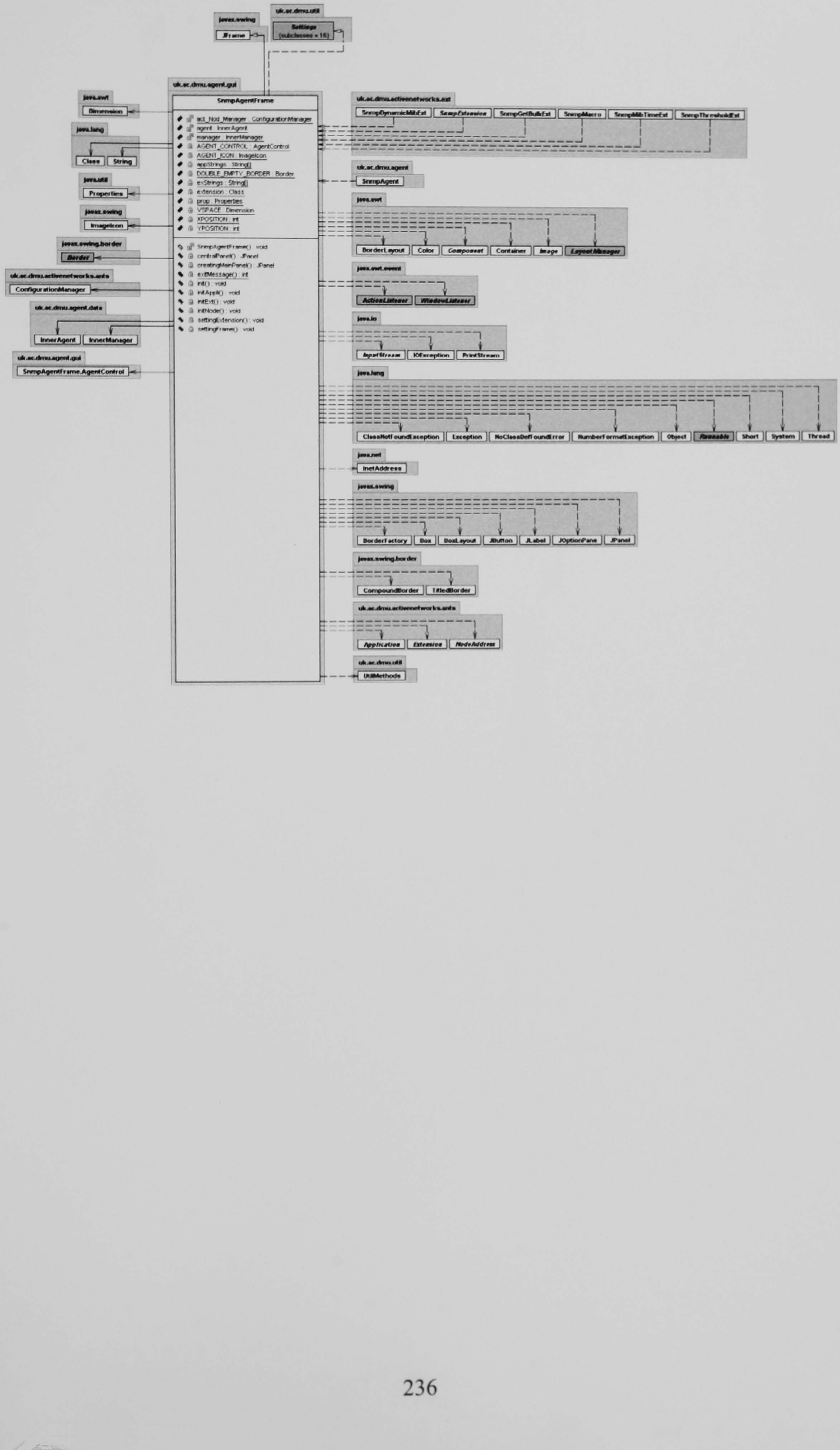
Class: InnerManagerException



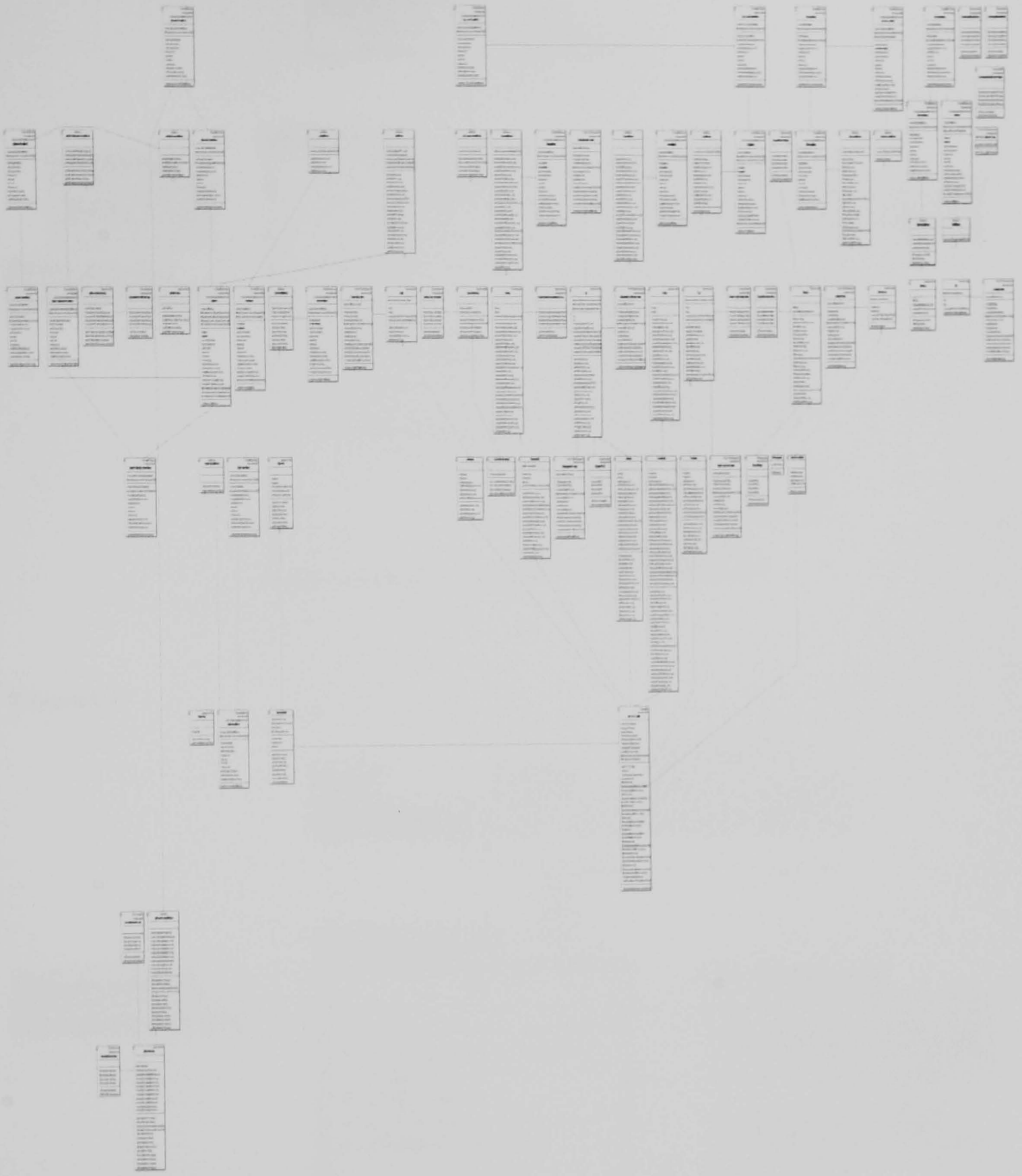
Package: uk.ac.dmu.agent.gui



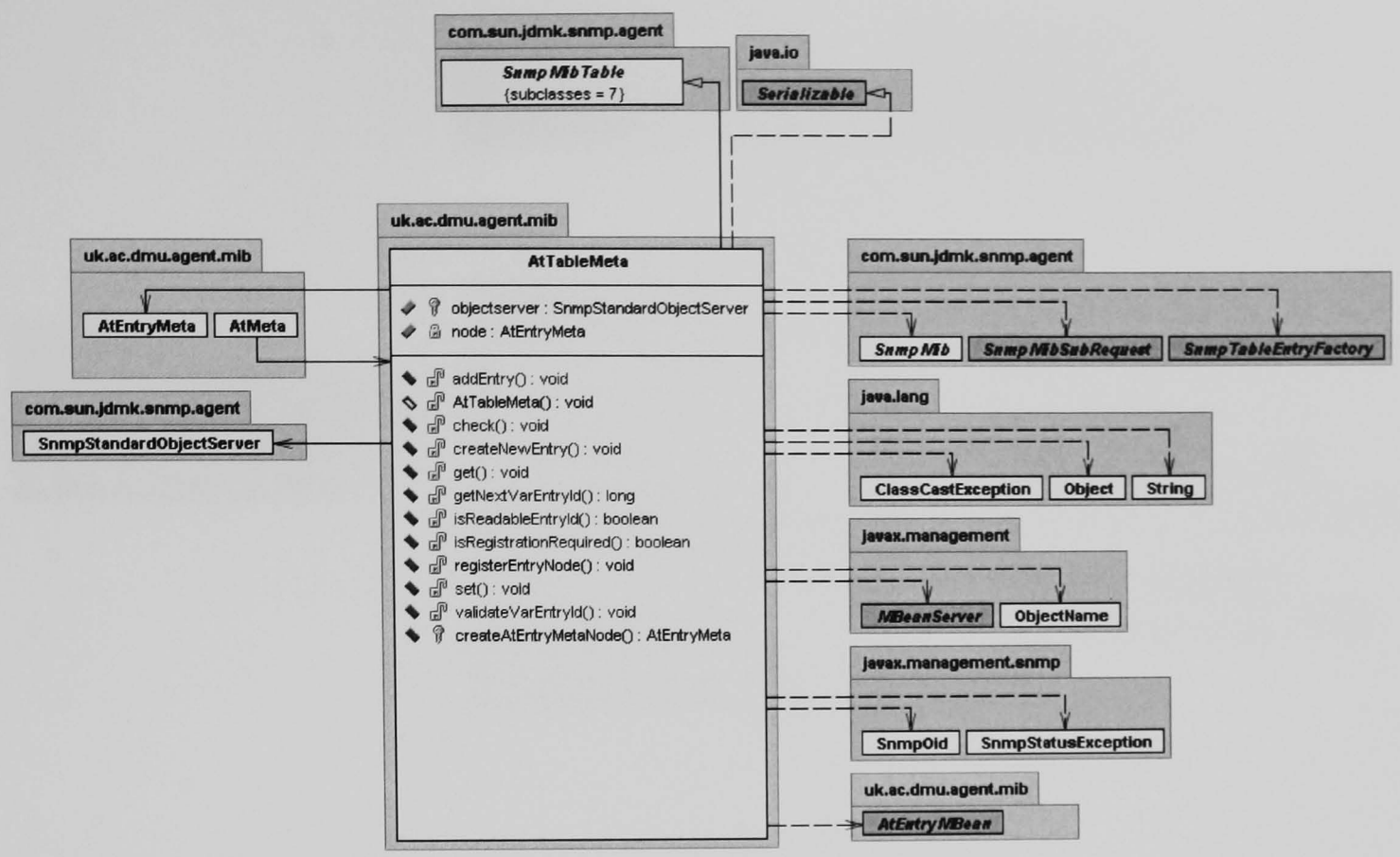
Class: SnmpAgentFrame



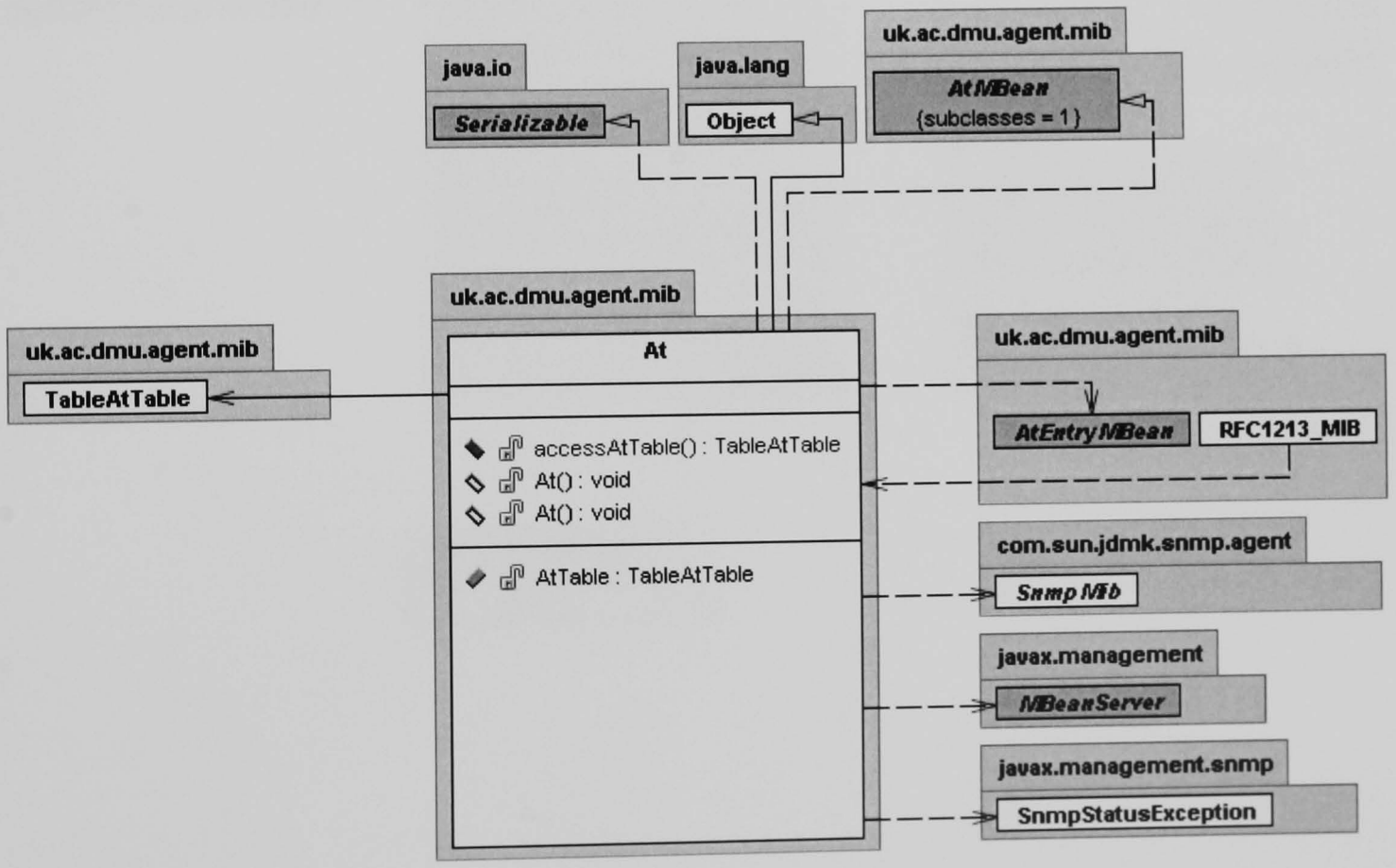
Package uk.ac.dmu.agent.mib



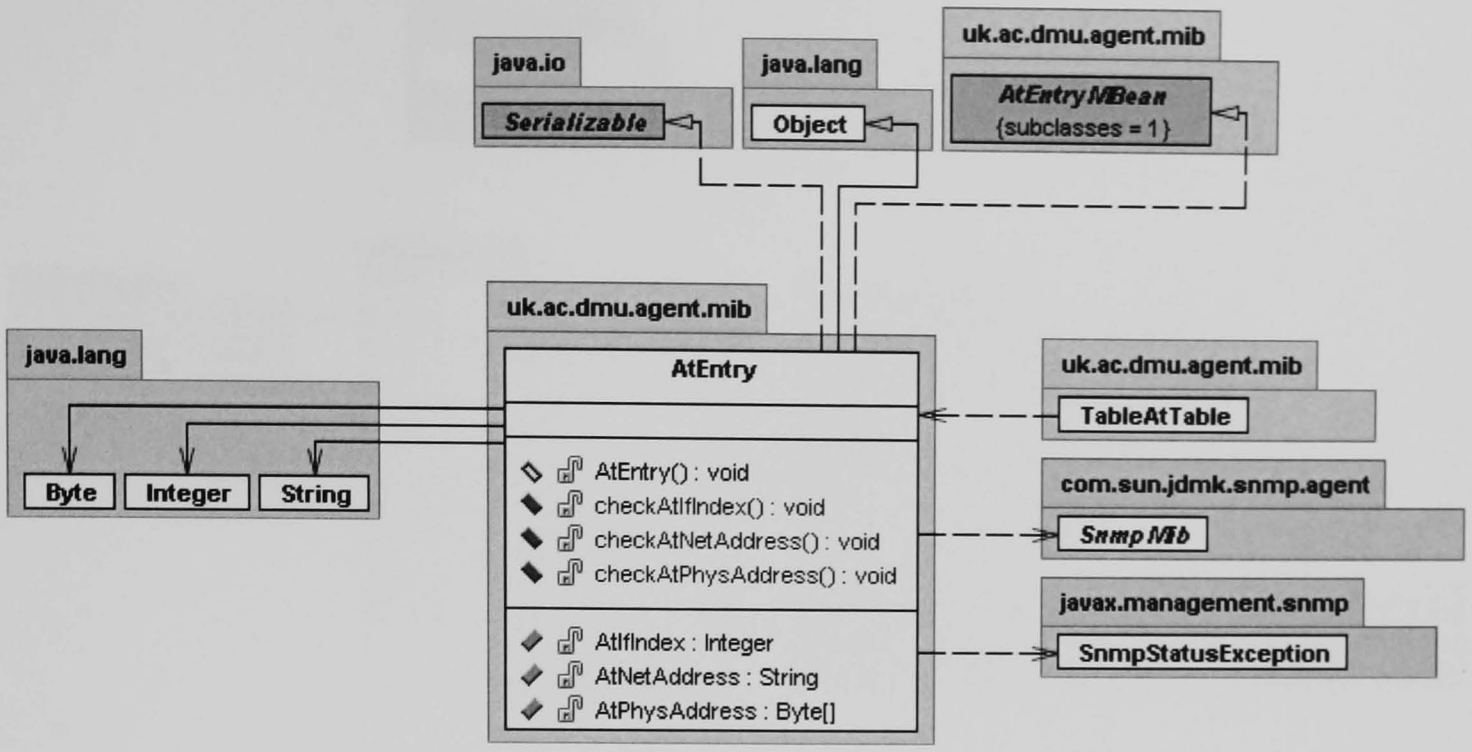
Class: AtTableMeta



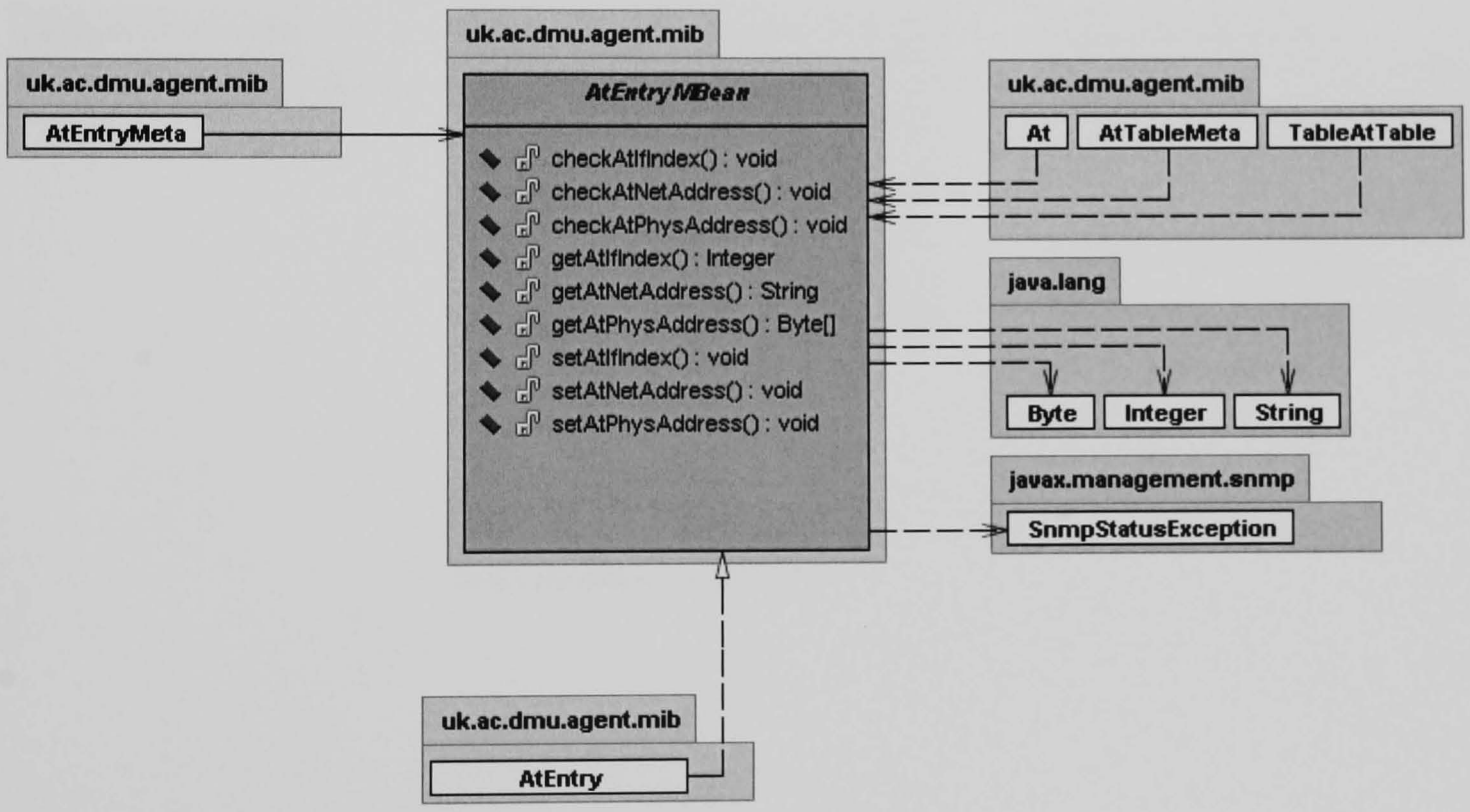
Class:AT



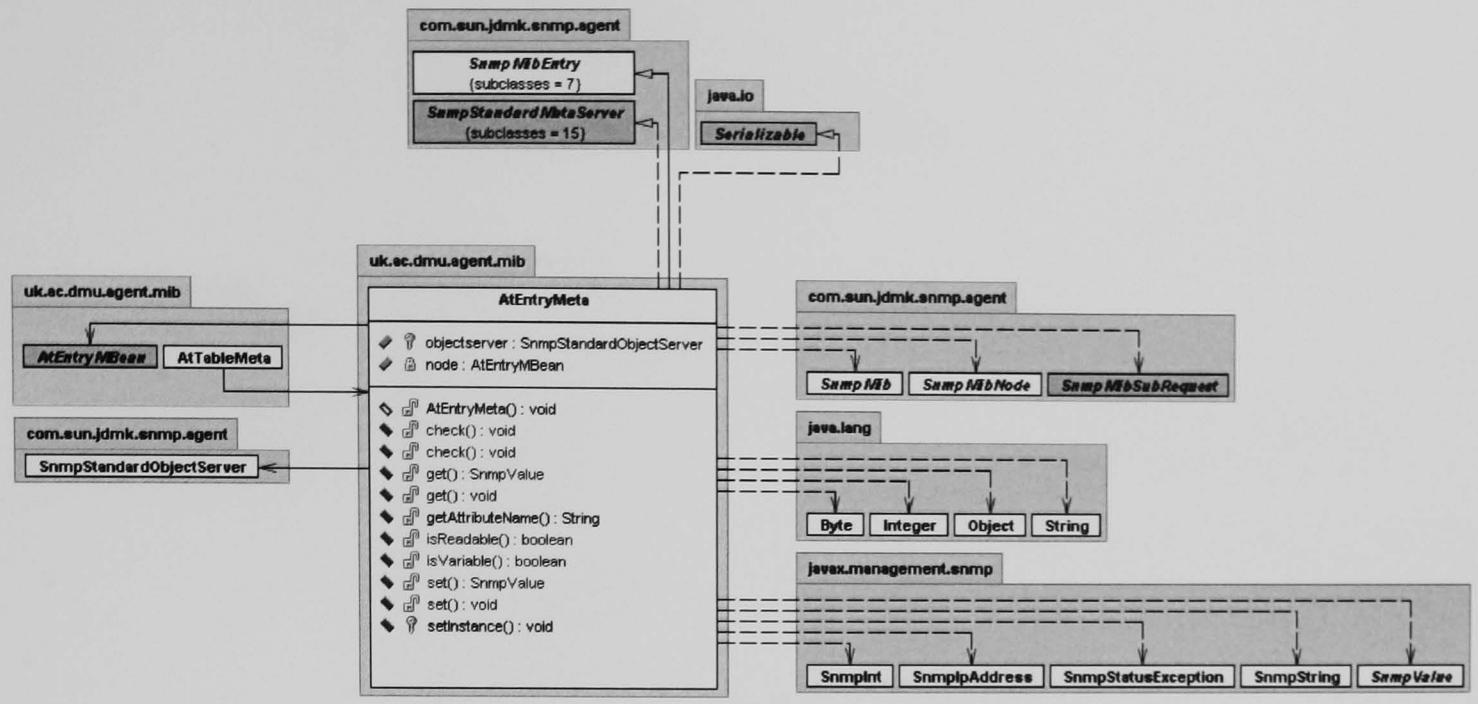
Class: AtEntry



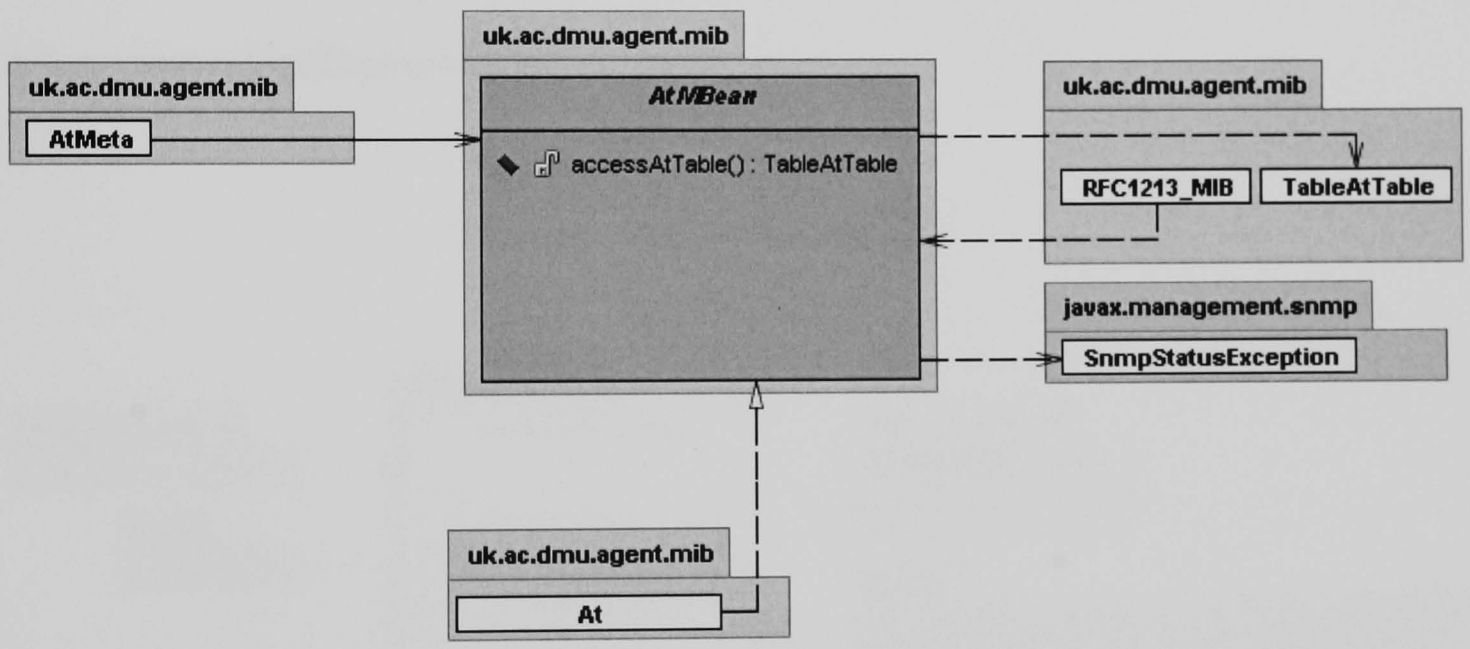
Interface AtEntryMBean



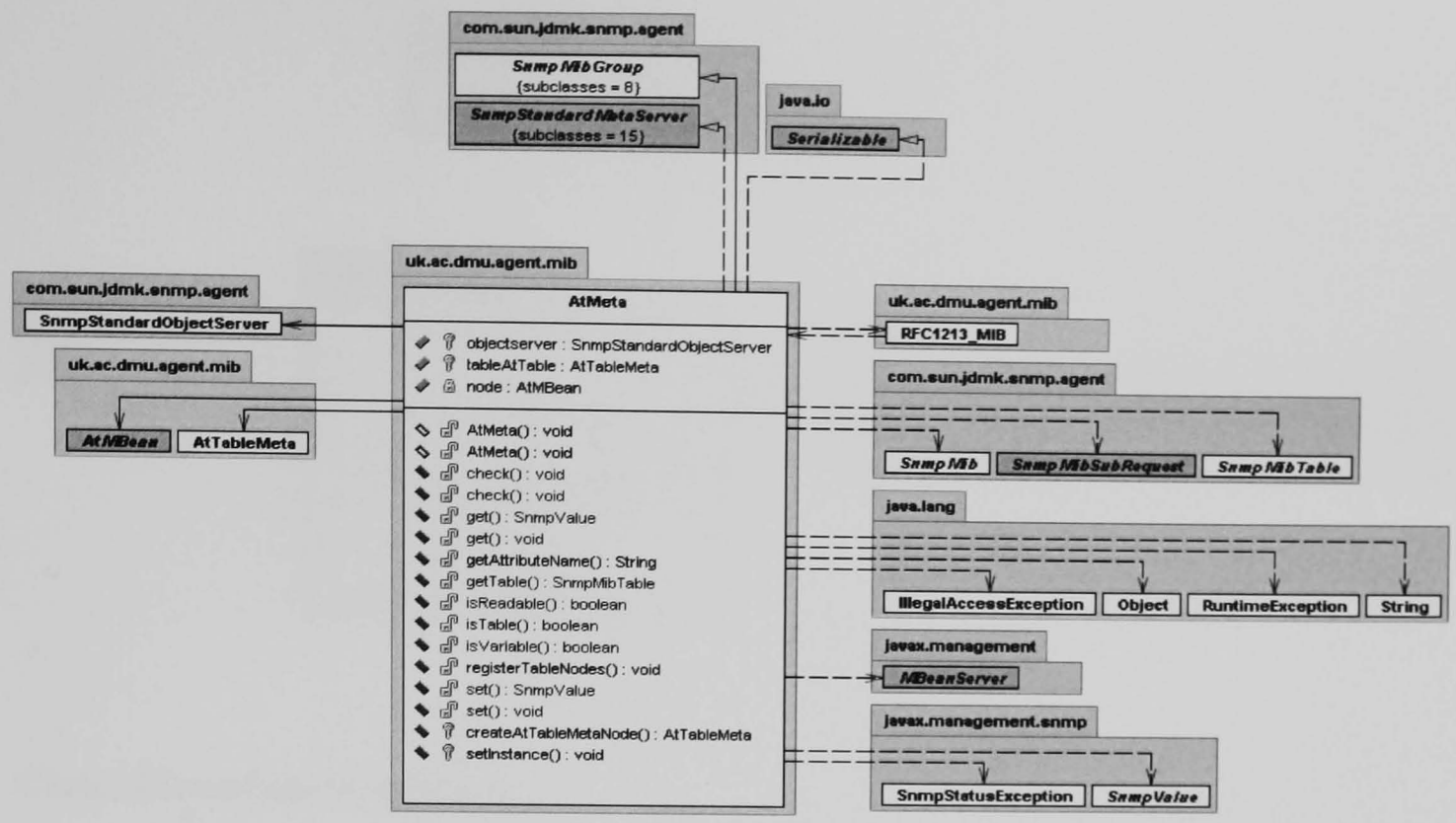
Class:AtEntryMeta



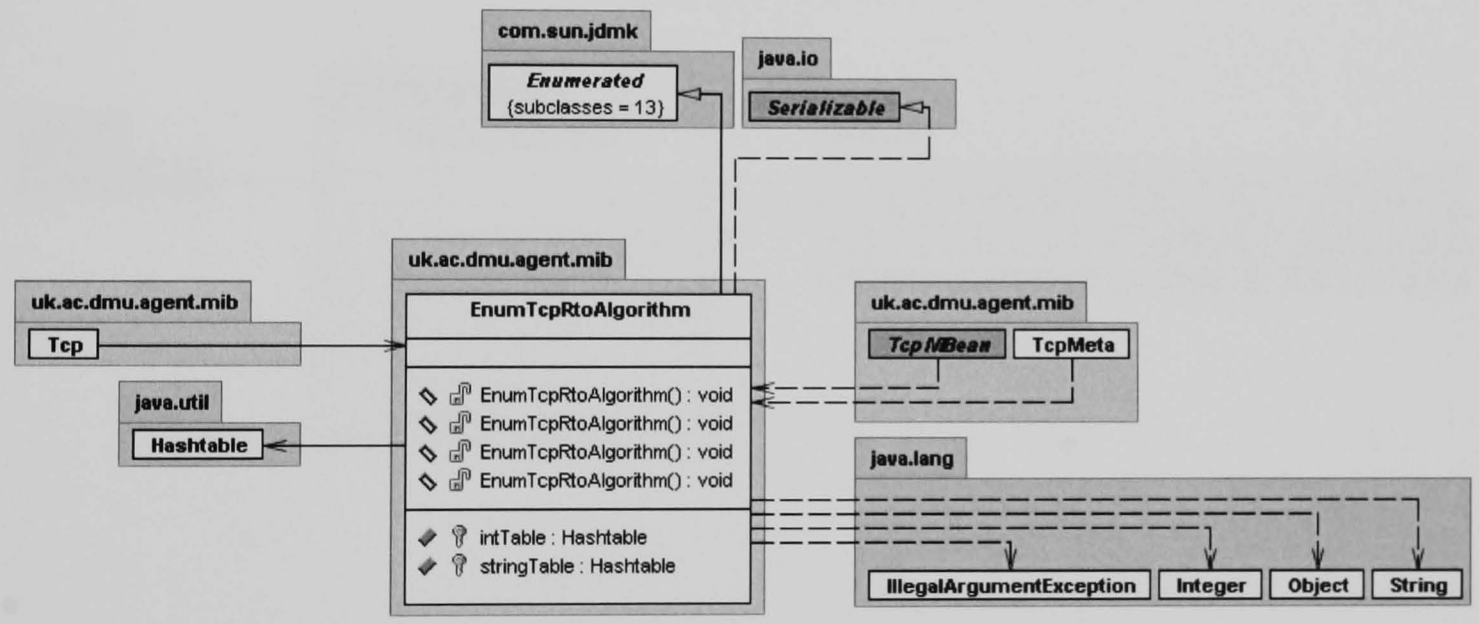
Interface: AtMetaBean



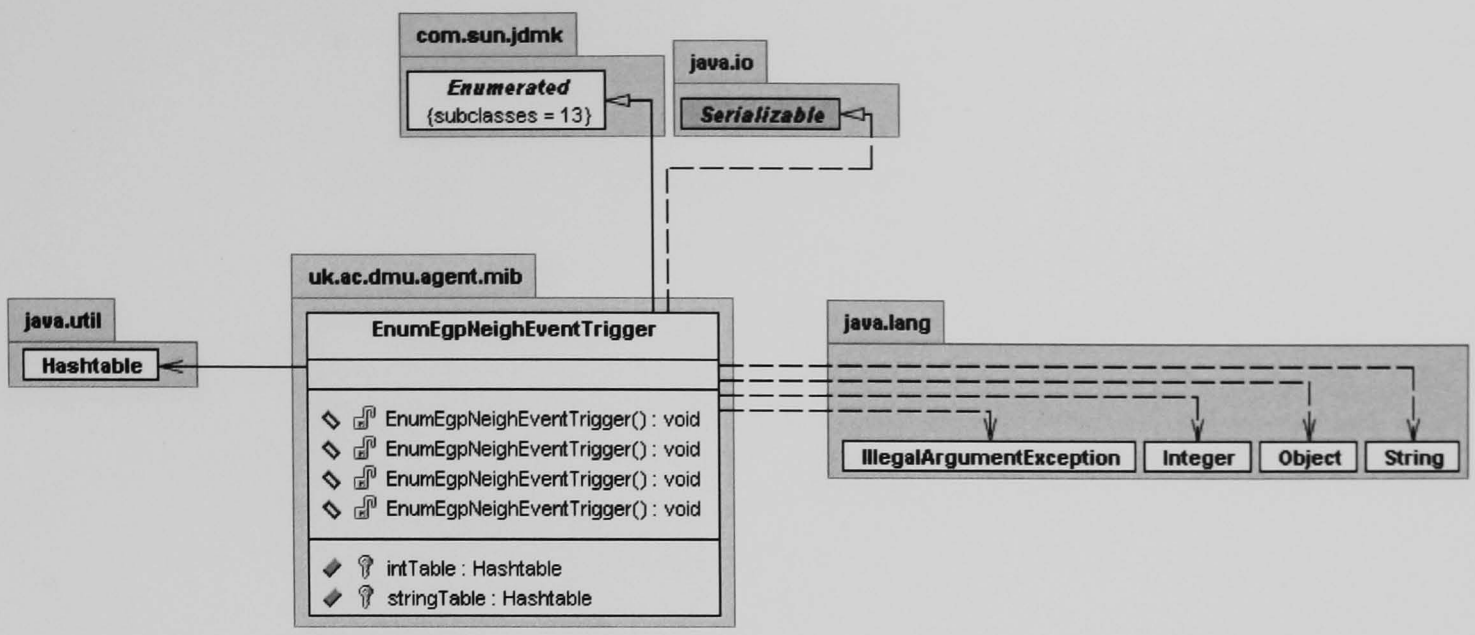
Class: AtMeta



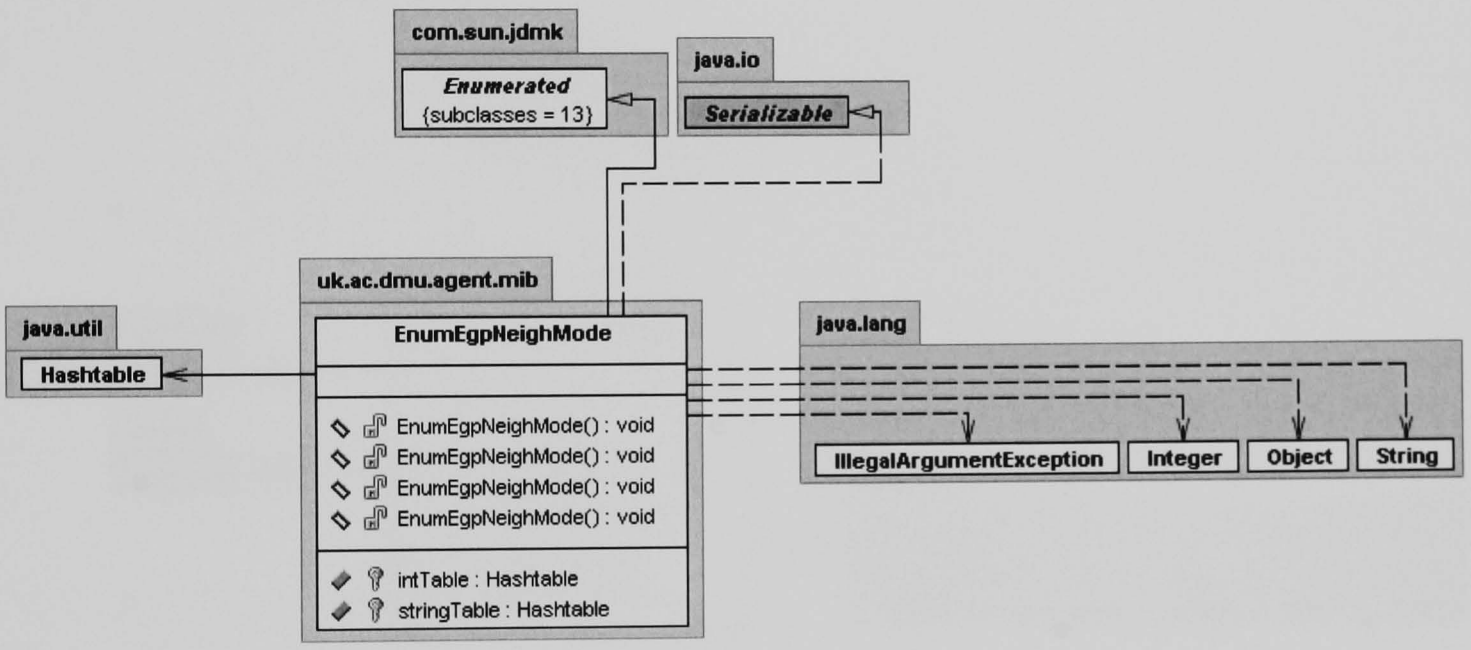
Class: EnumTcpRtoAlgorithm



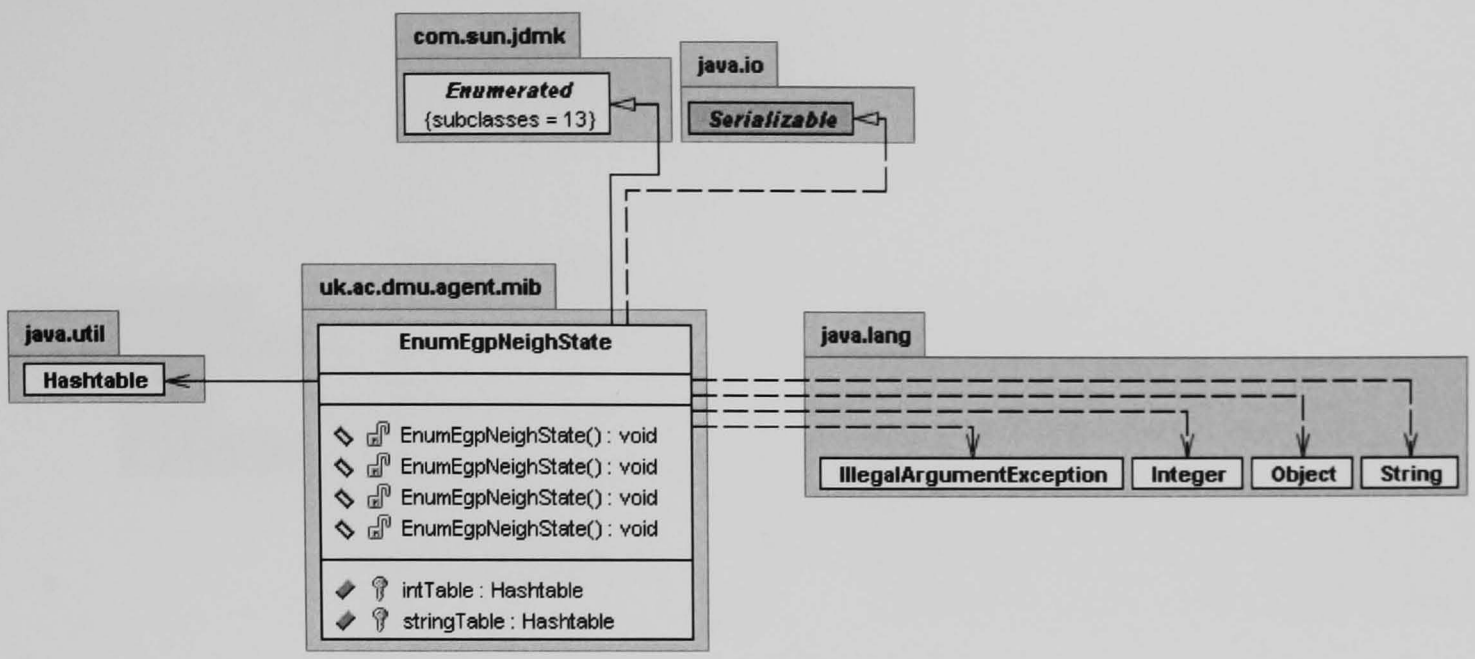
Class: EnumEgpNeighEventTrigger



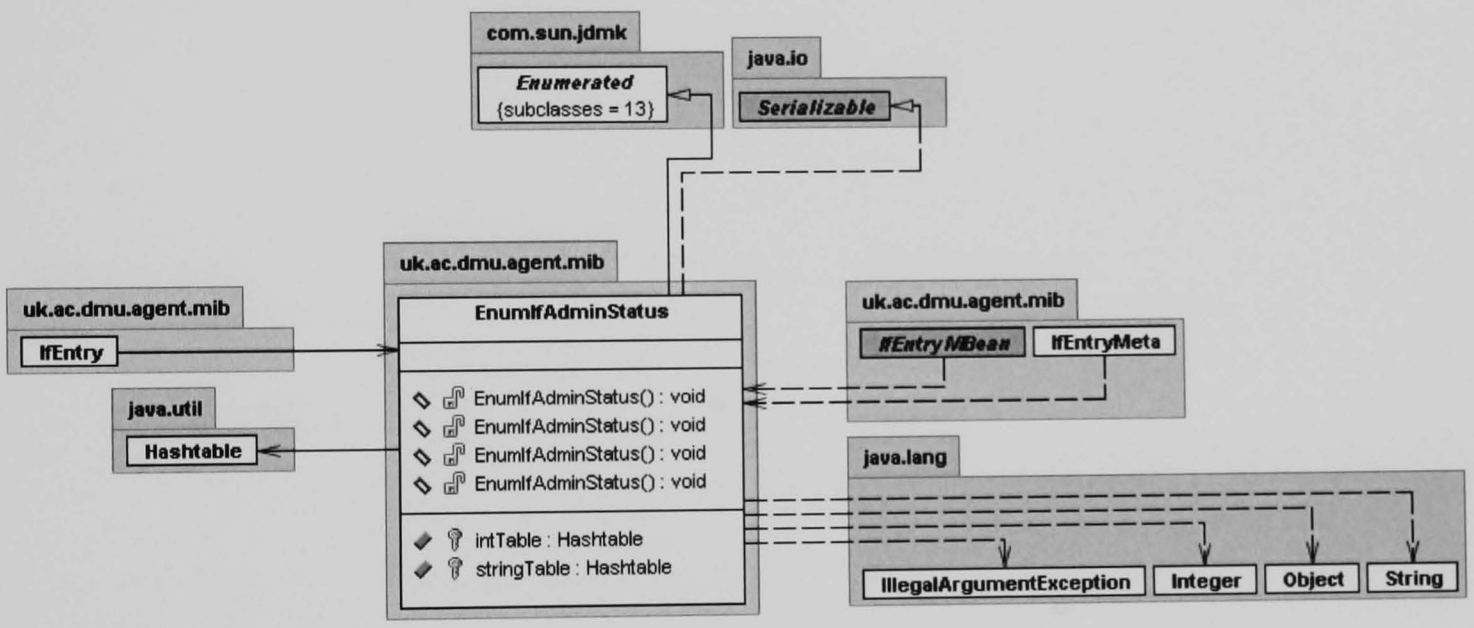
Class: EnumEgpNeighMode



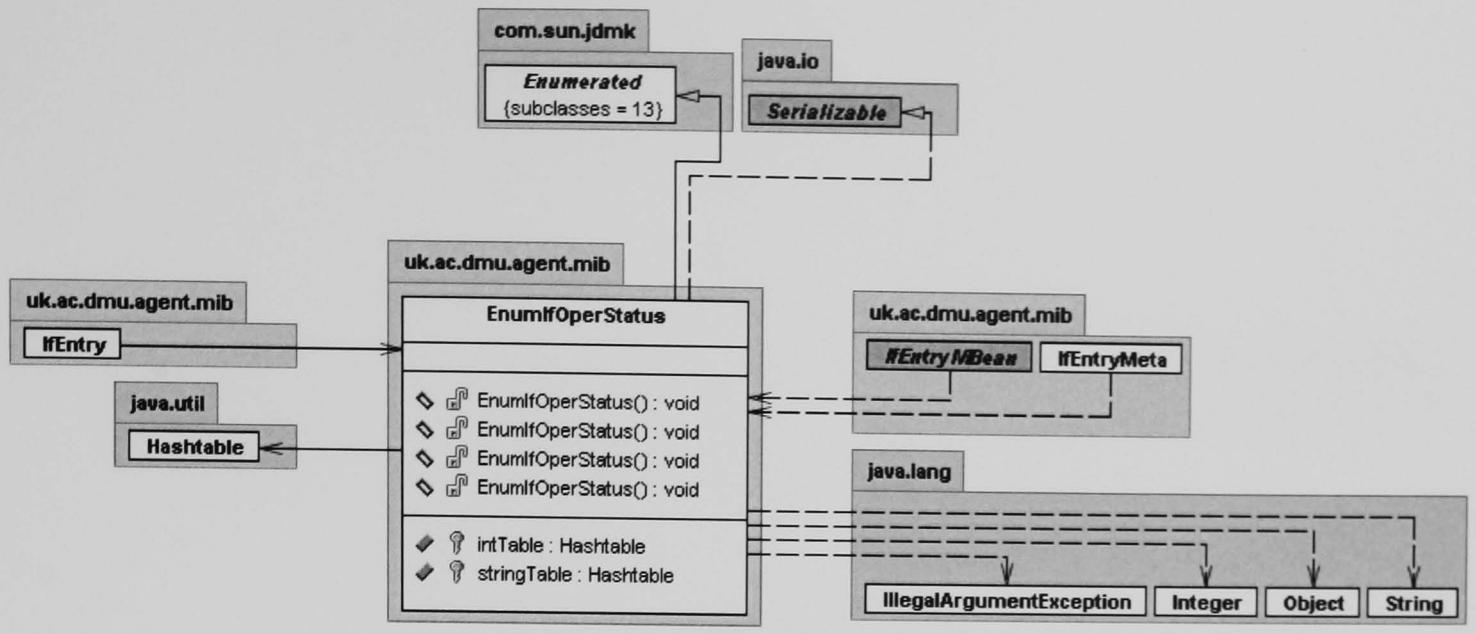
Class: EnumEgpNeighState



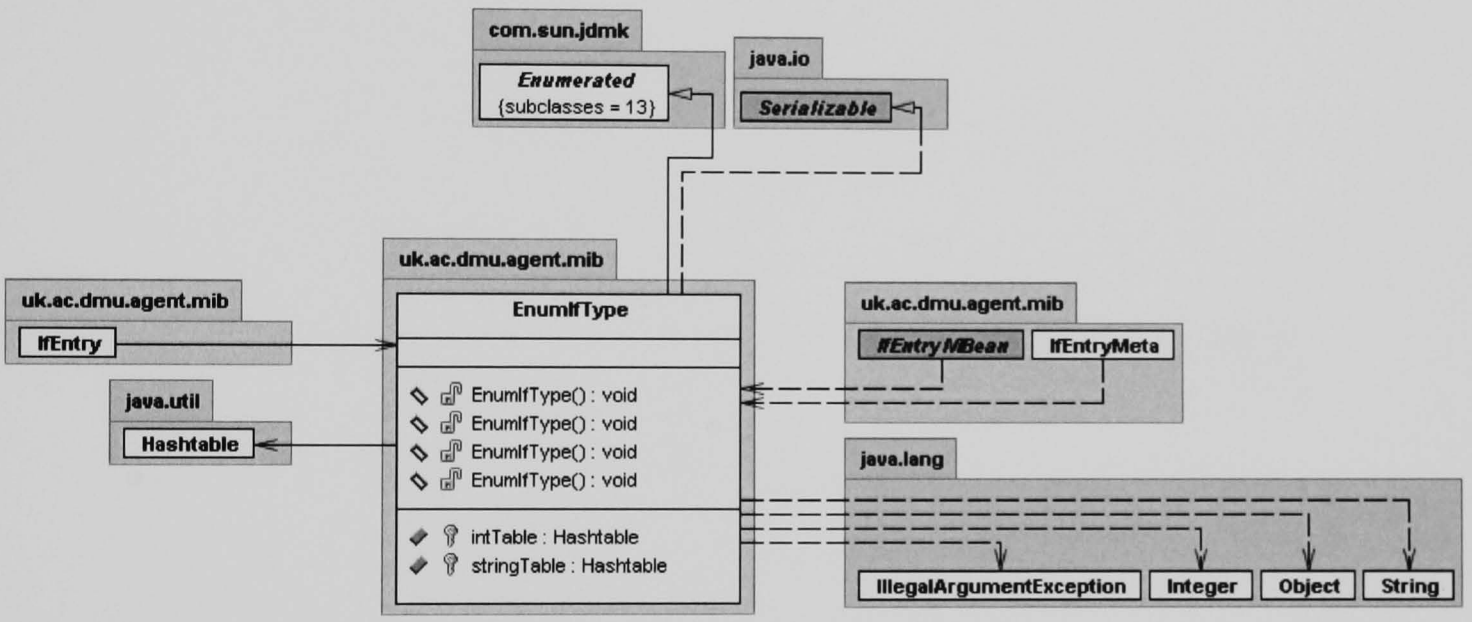
Class: EnumIfAdminStatus



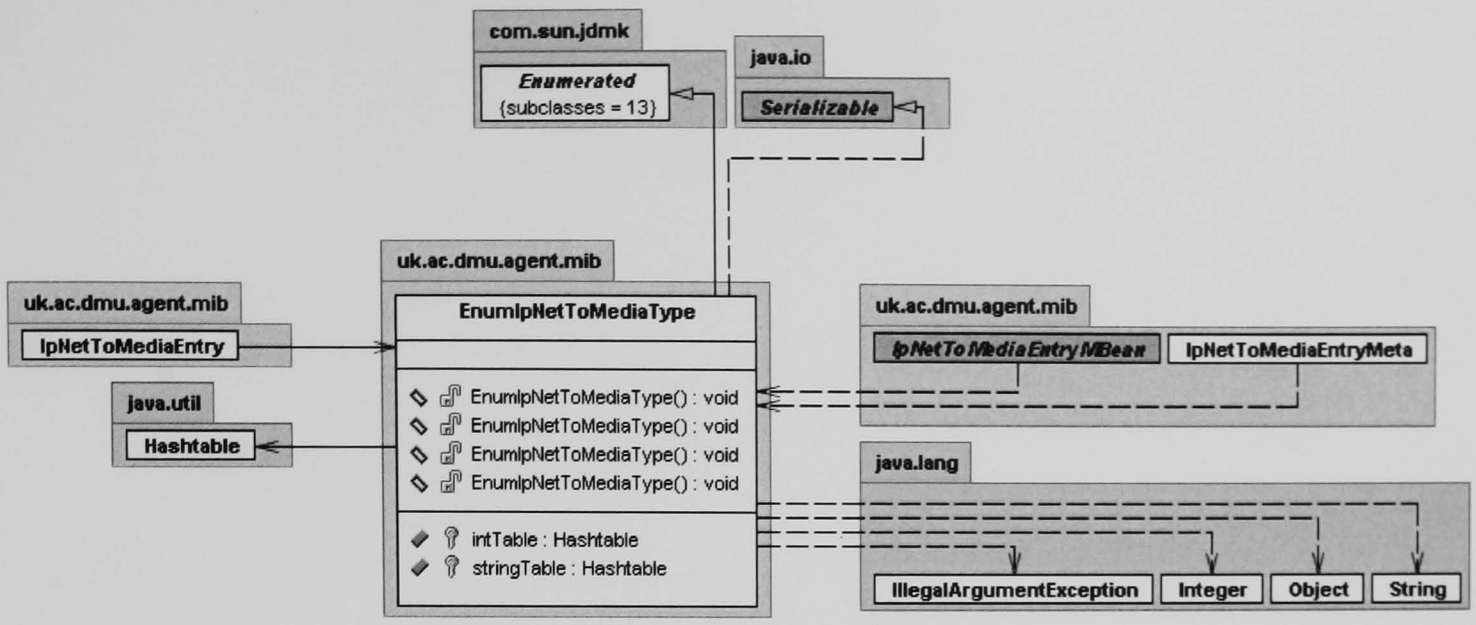
Class: EnumIfOperStatus



Class:EnumIfType

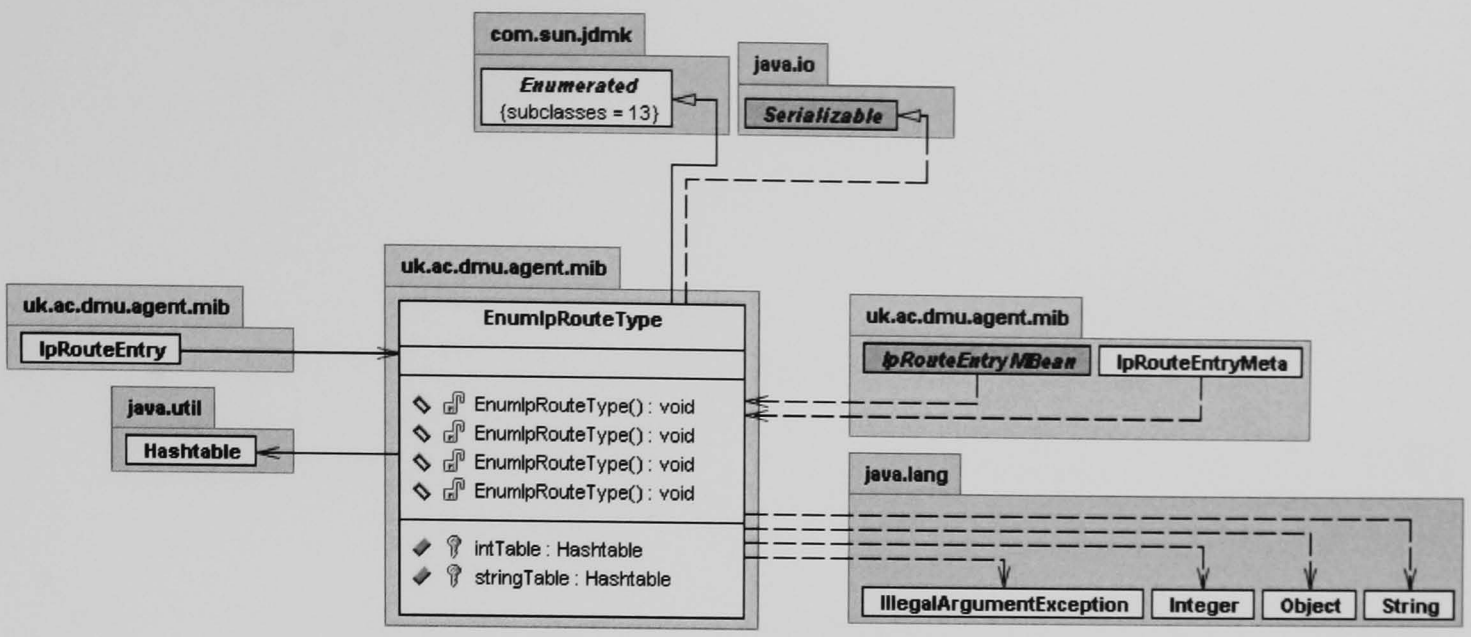


Class: EnumIpNetToMediaType

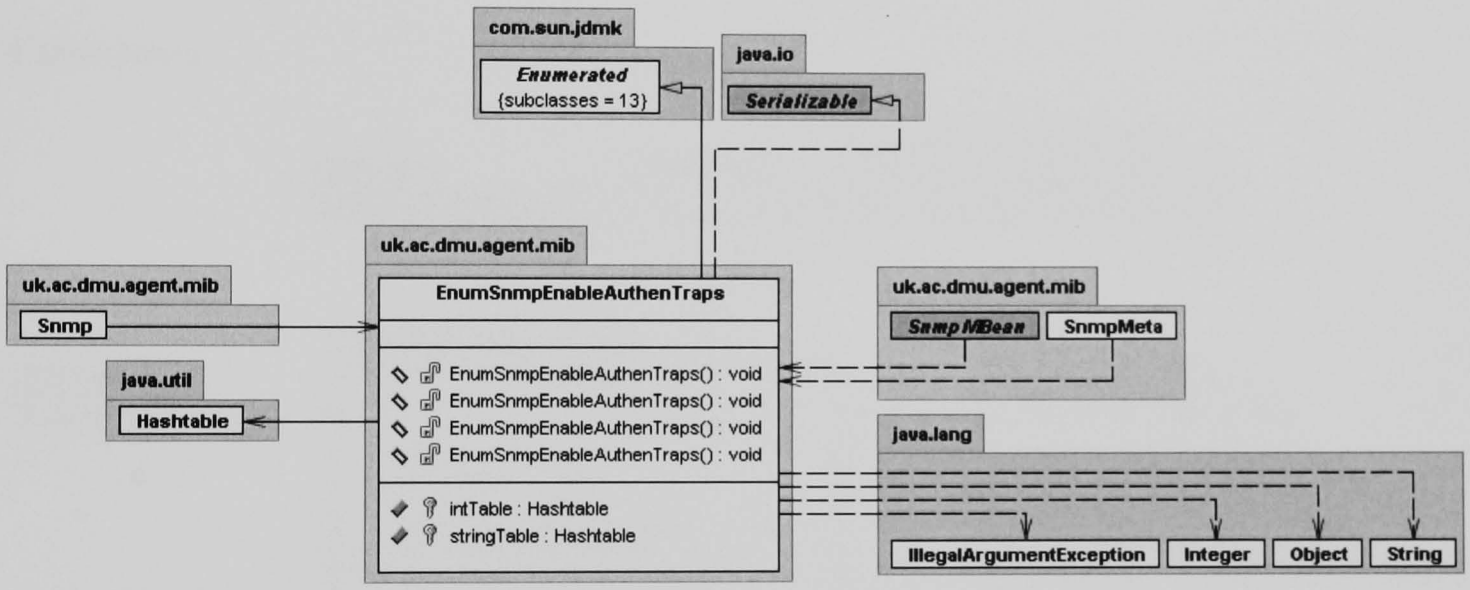


Class: EnumIpRouteProto

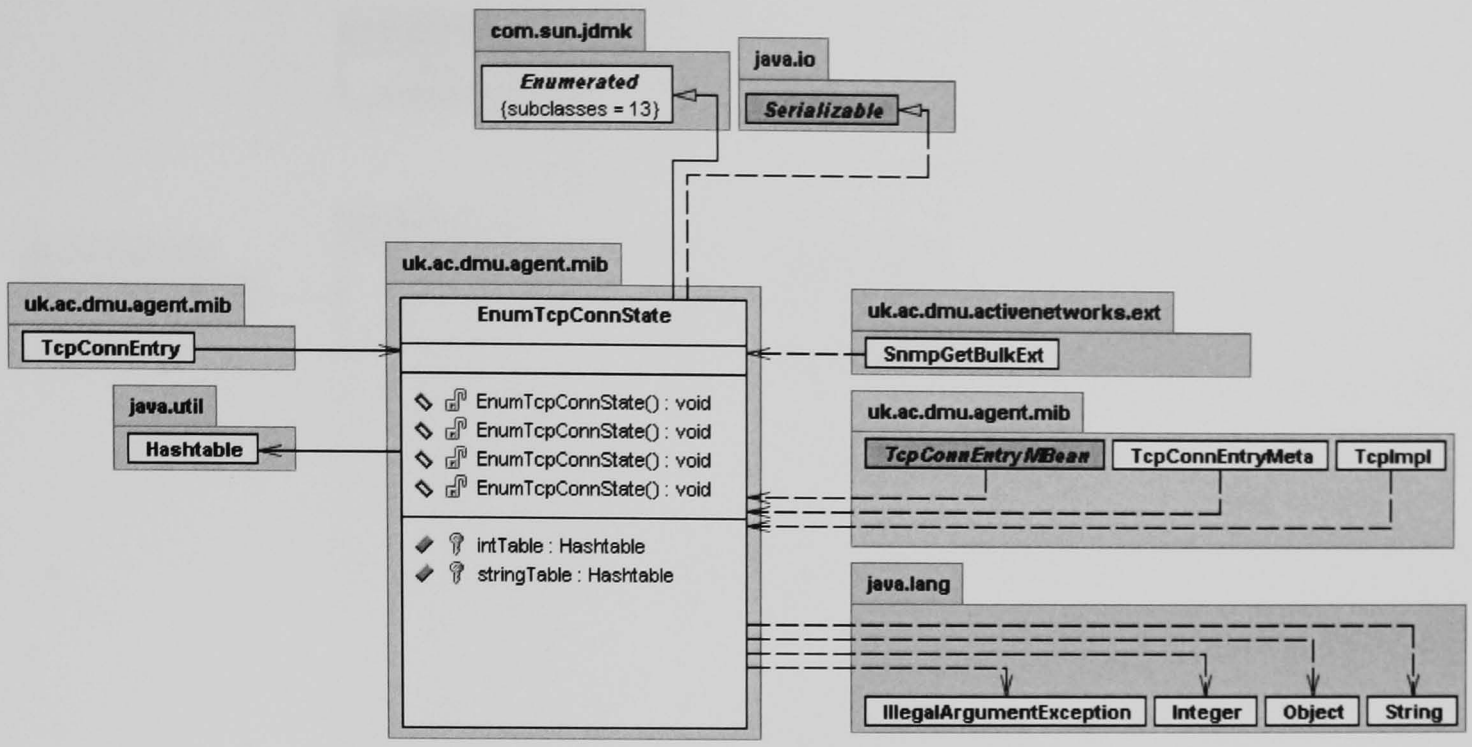
Class: EnumIpRouteType



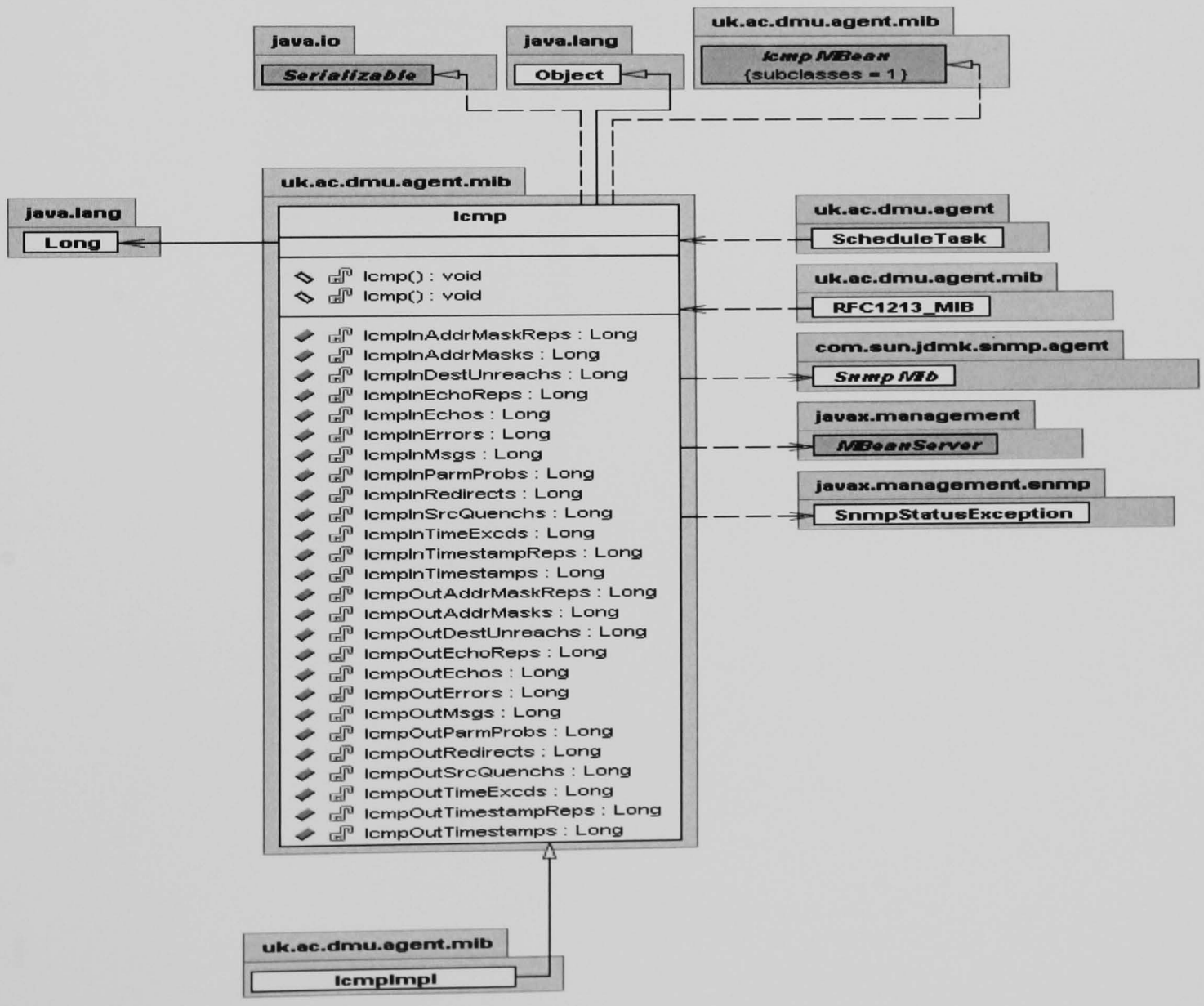
Class: EnumSnmpEnableAuthenTraps



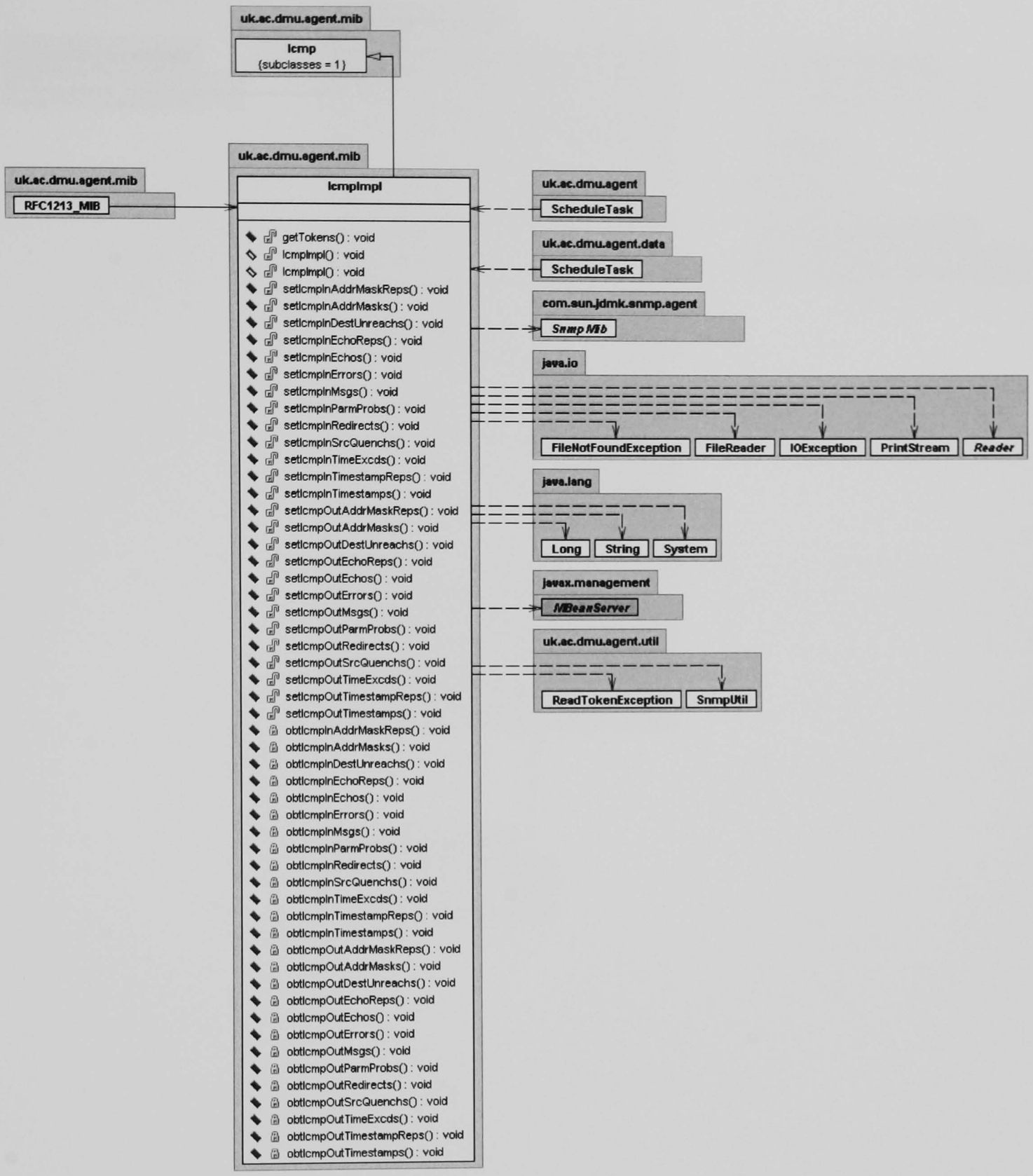
Class: EnumTcpConnState



Class Icmp



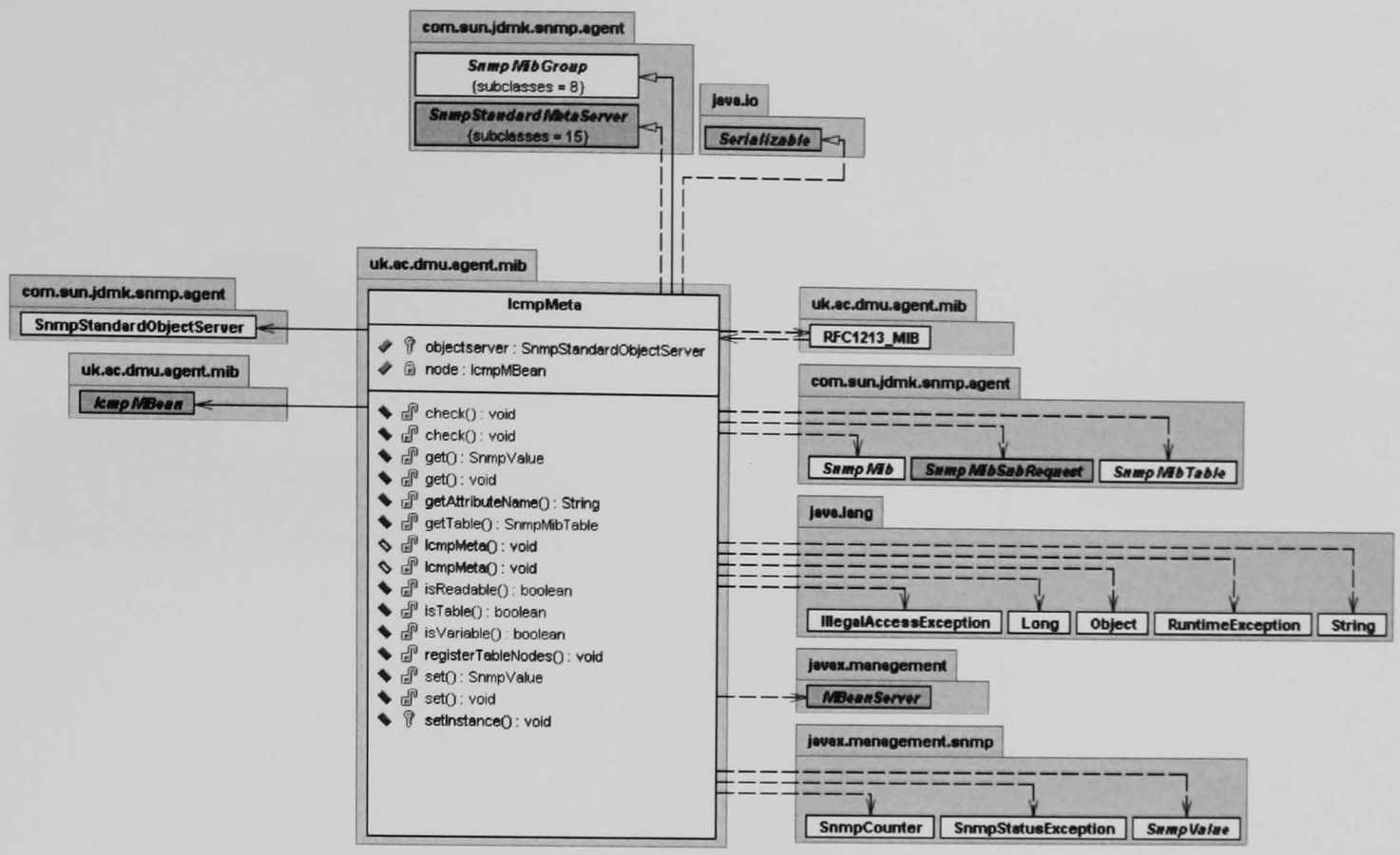
Class: IcmpImpl



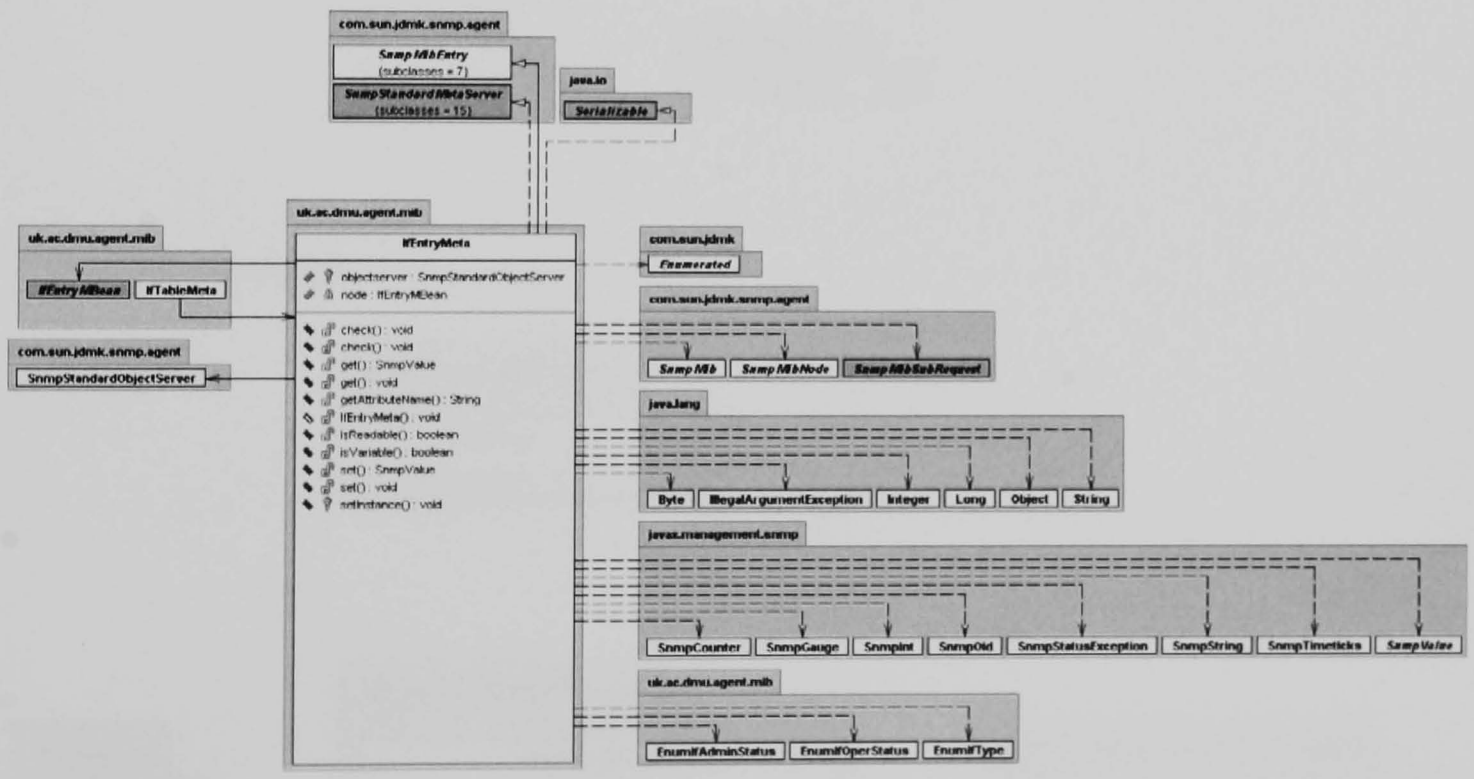
I

Interface: IcmpMBean

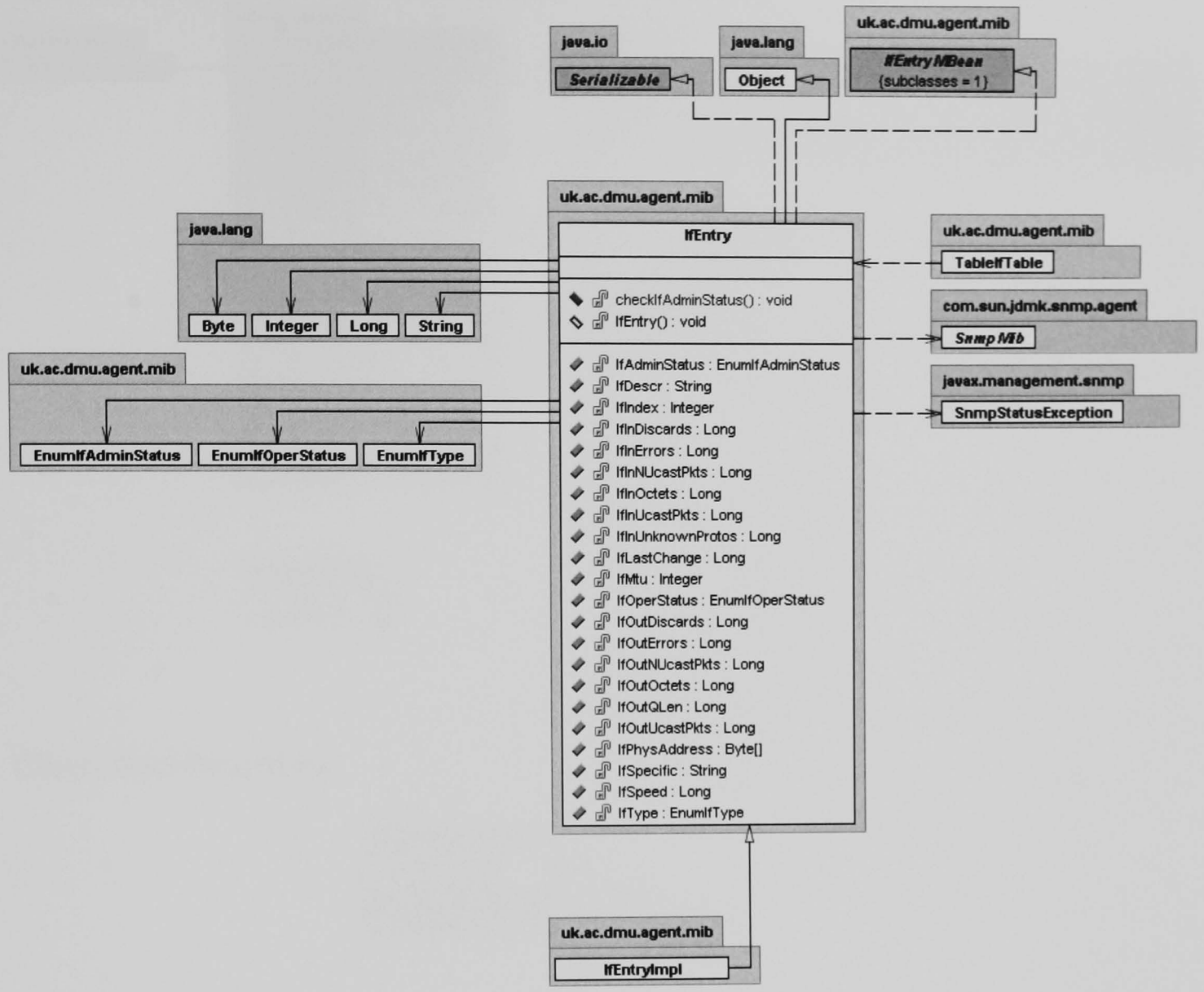
Class:IcmpMeta



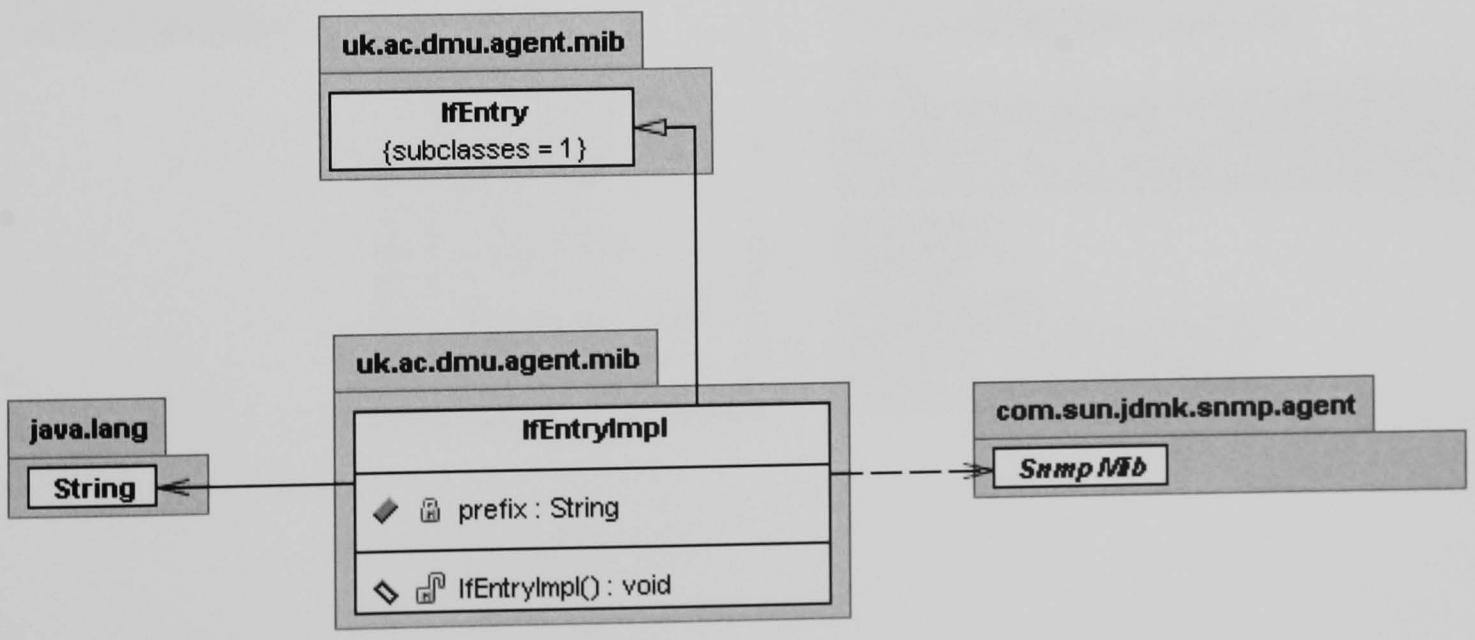
Class: IfEntryMeta



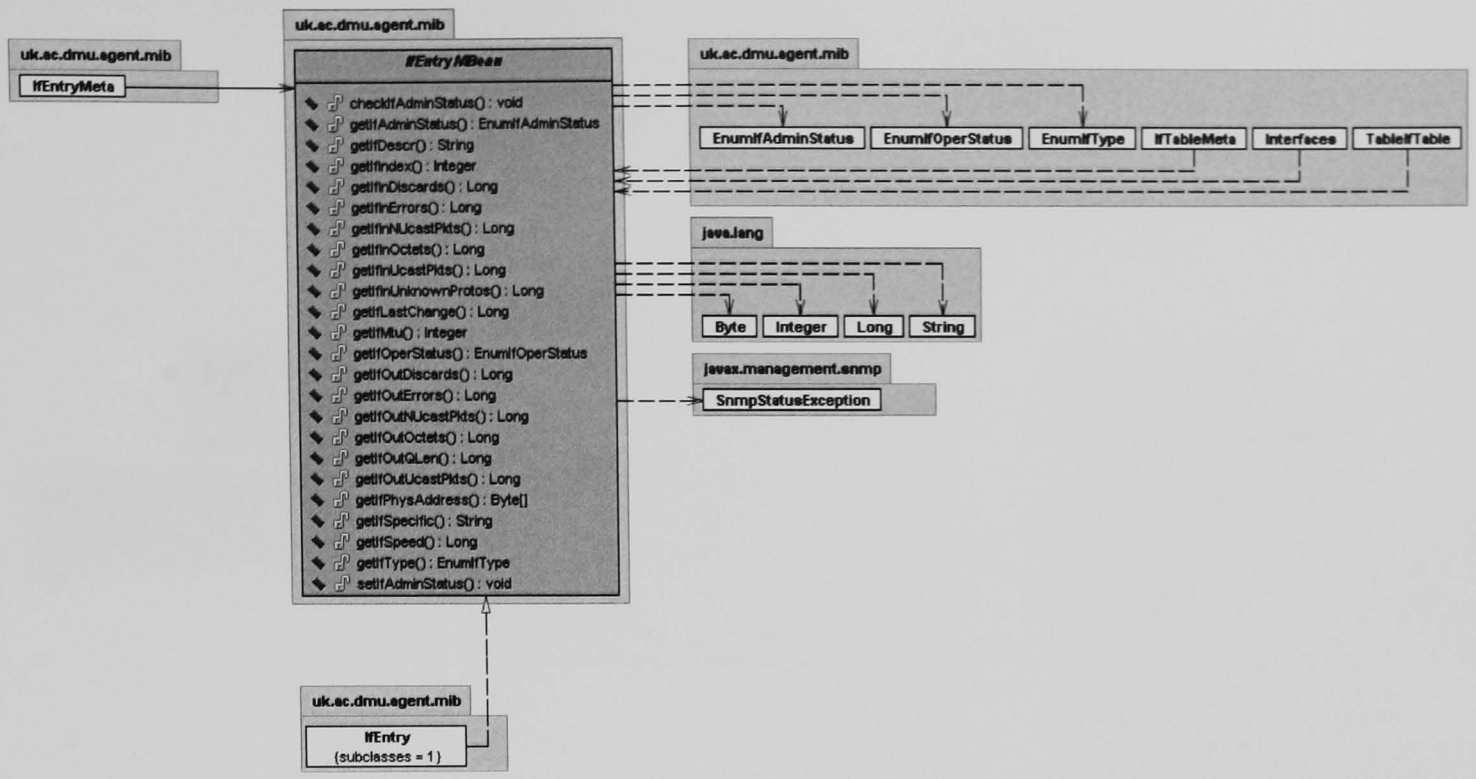
Class: IfEntry



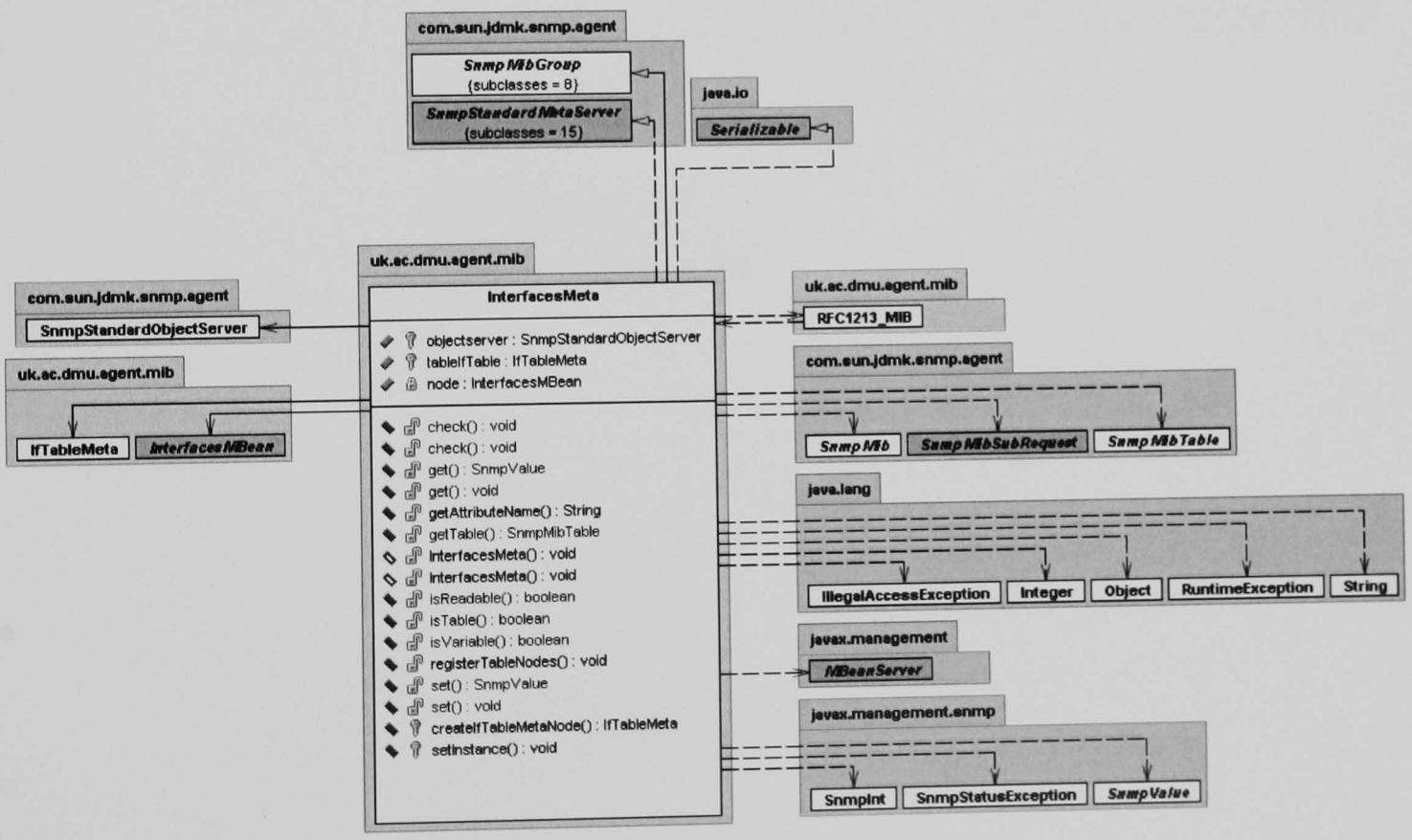
Class:IfEntryImpl



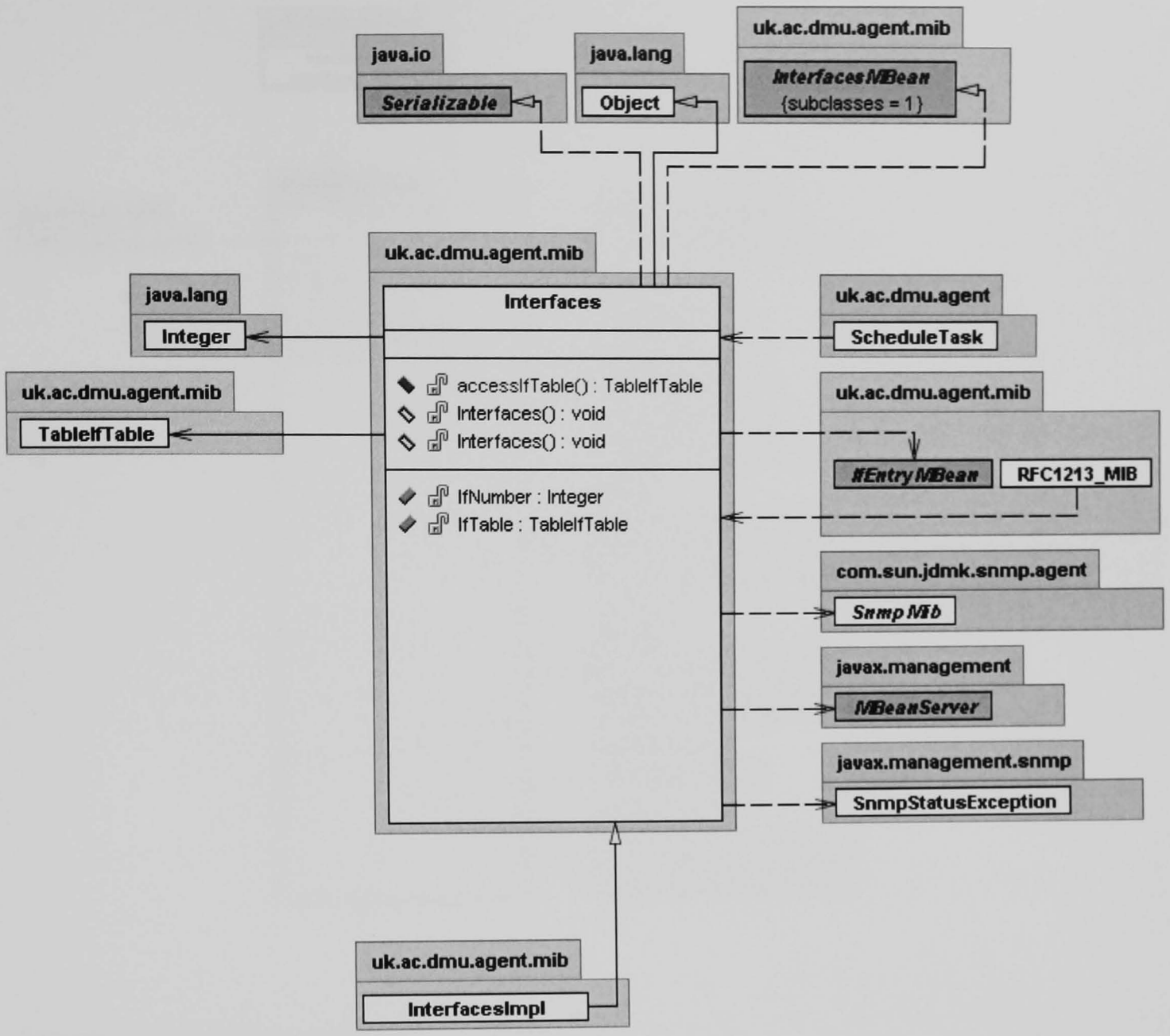
Interface:IfEntryMBean



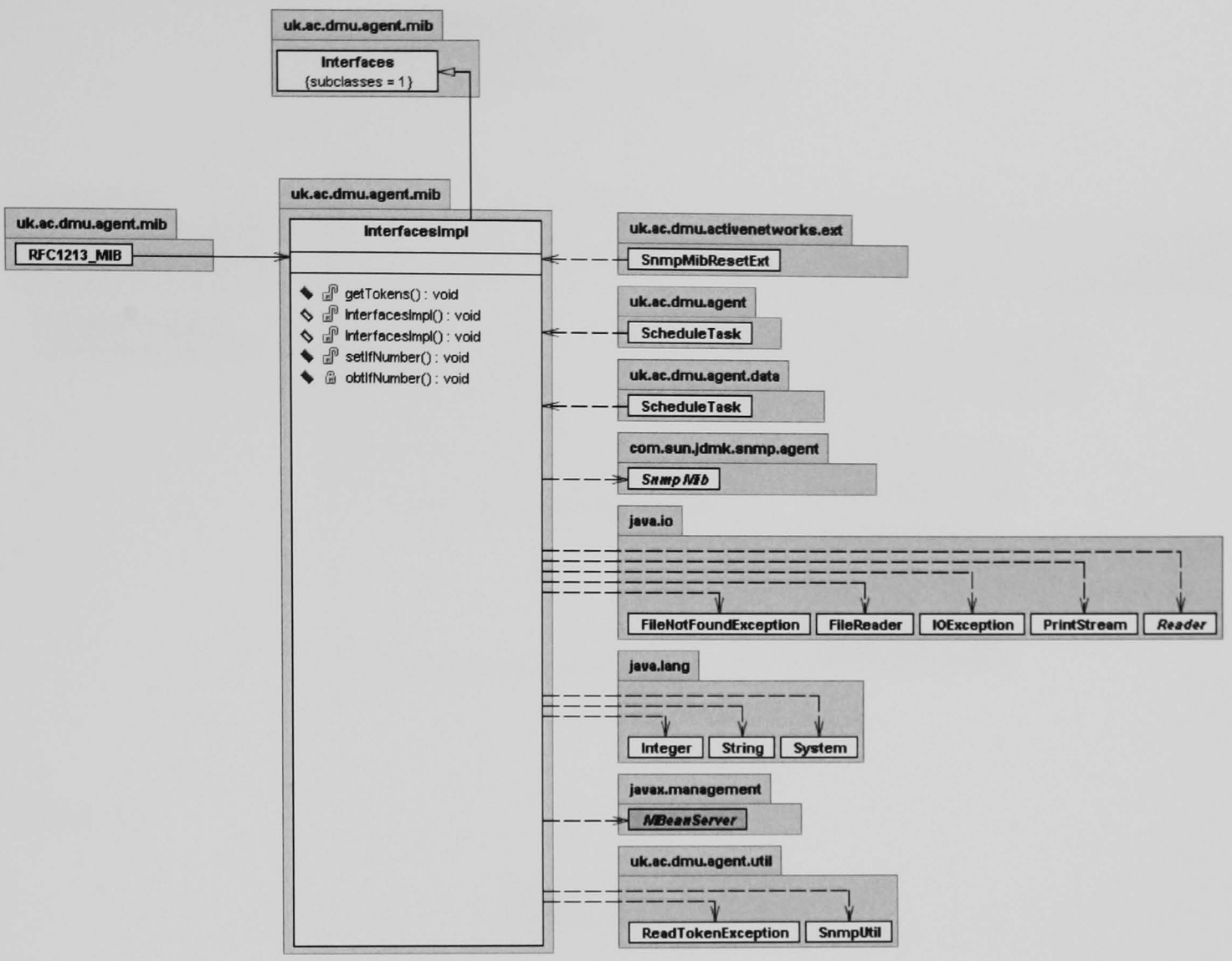
Class: InterfacesMeta



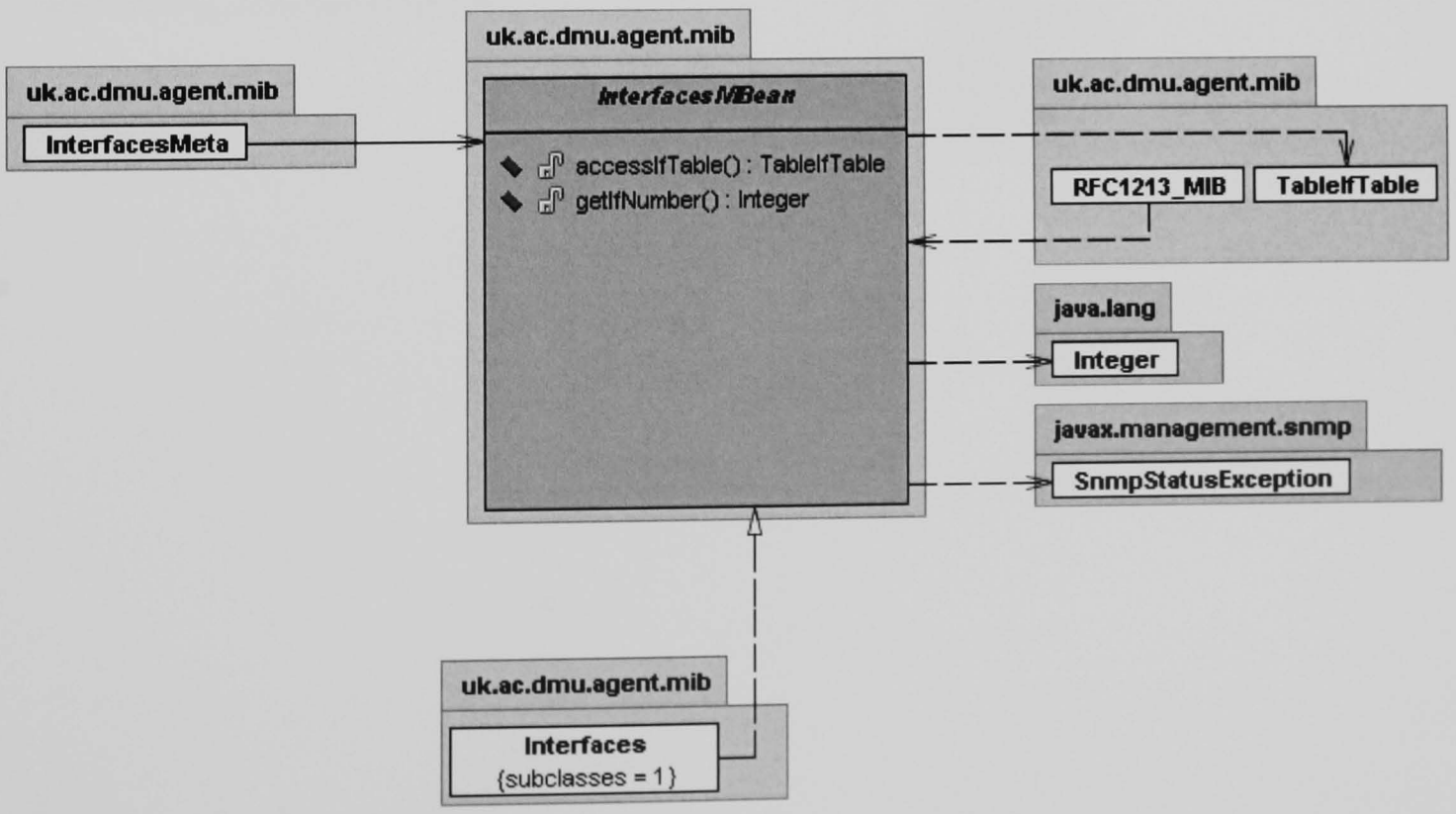
Class:Interfaces



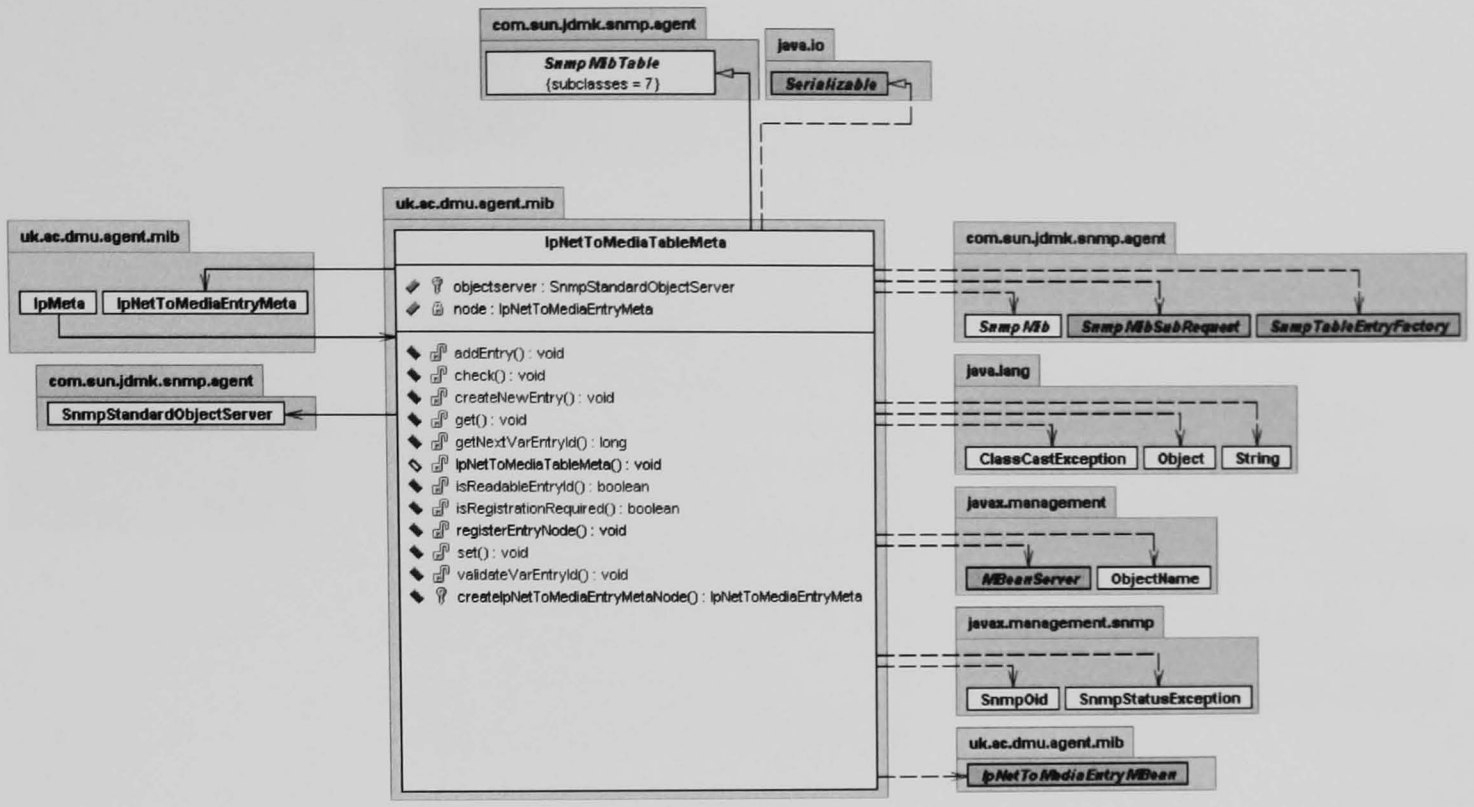
Class:InterfacesImpl



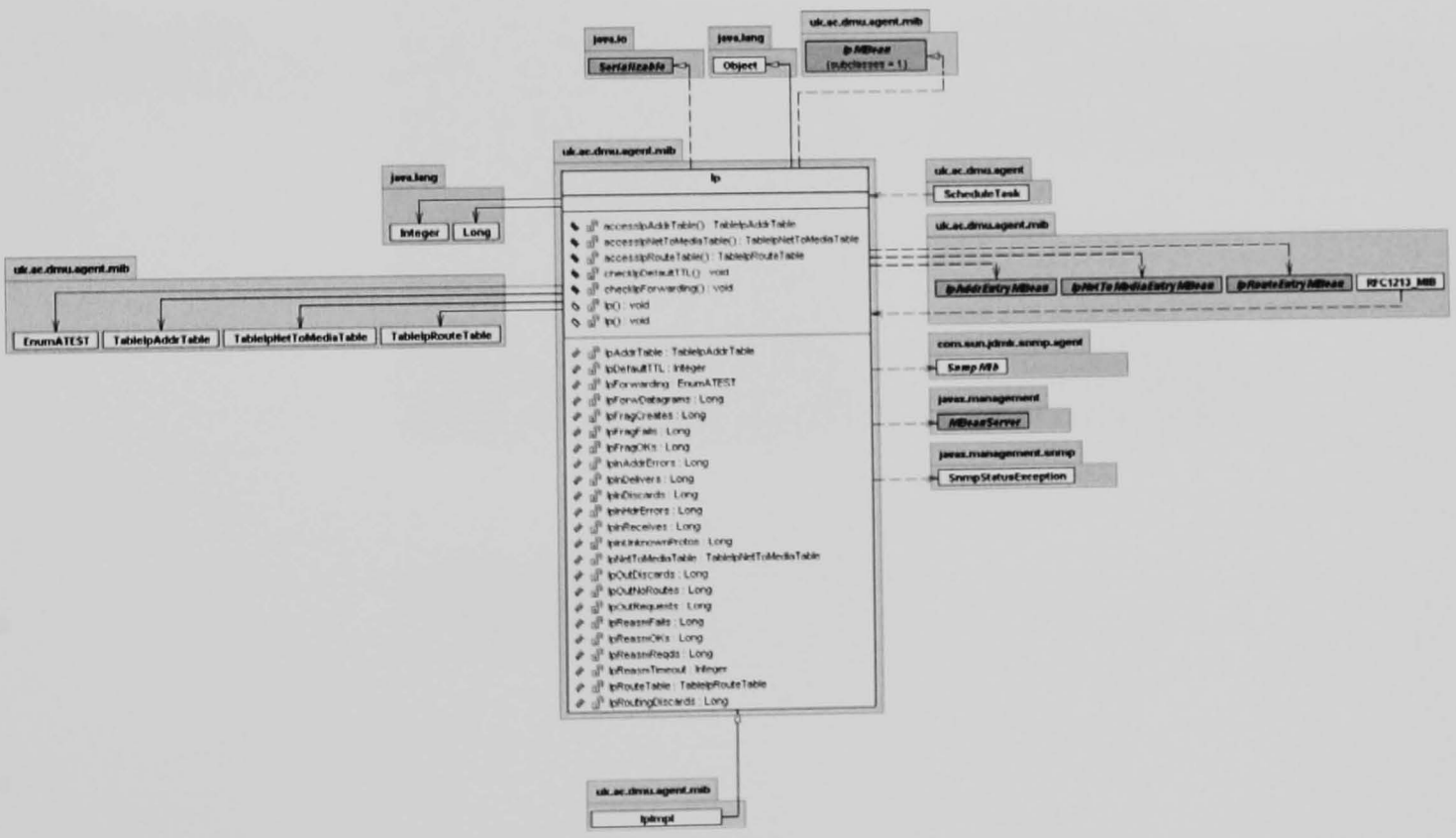
Interface: InterfacesMBean



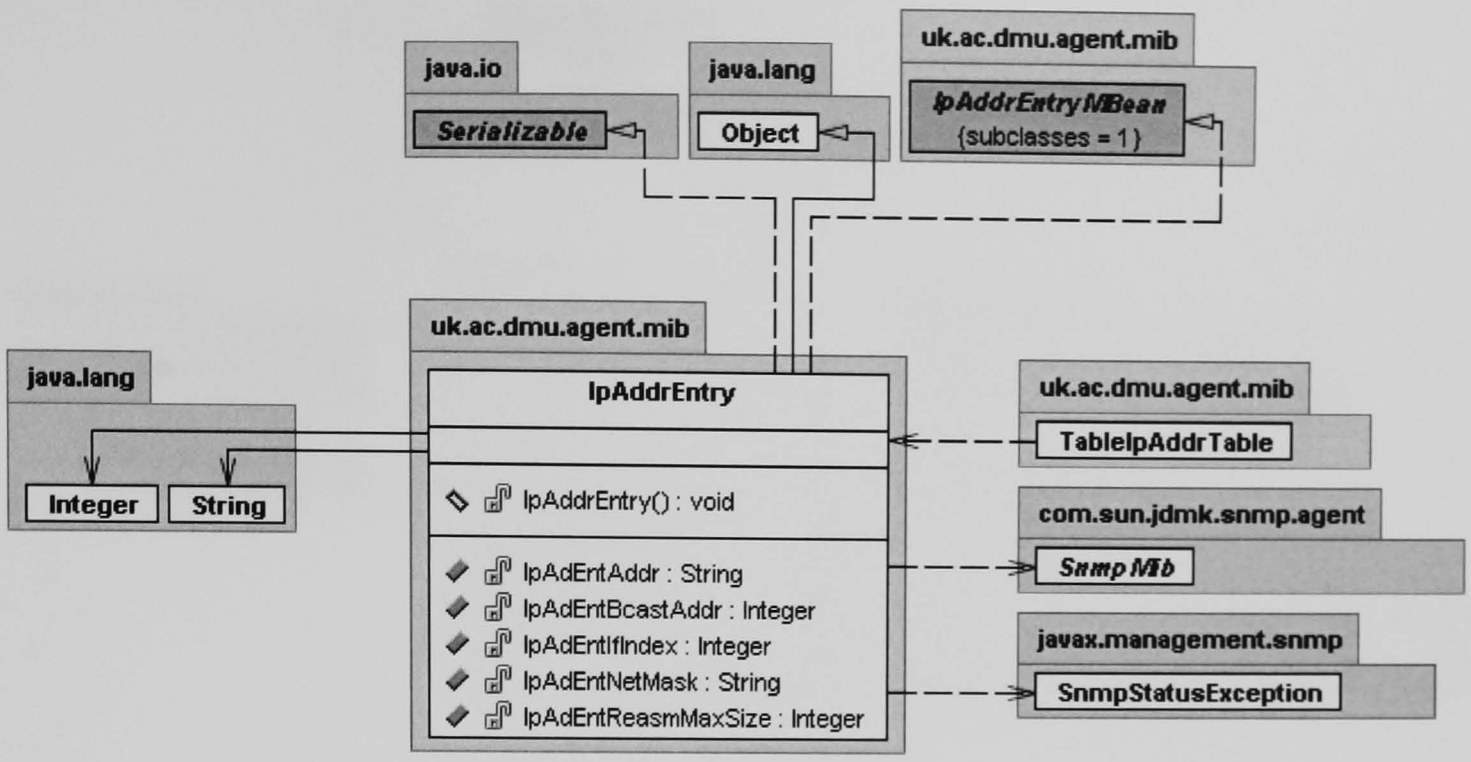
Class:IpNetToMediaTableMeta



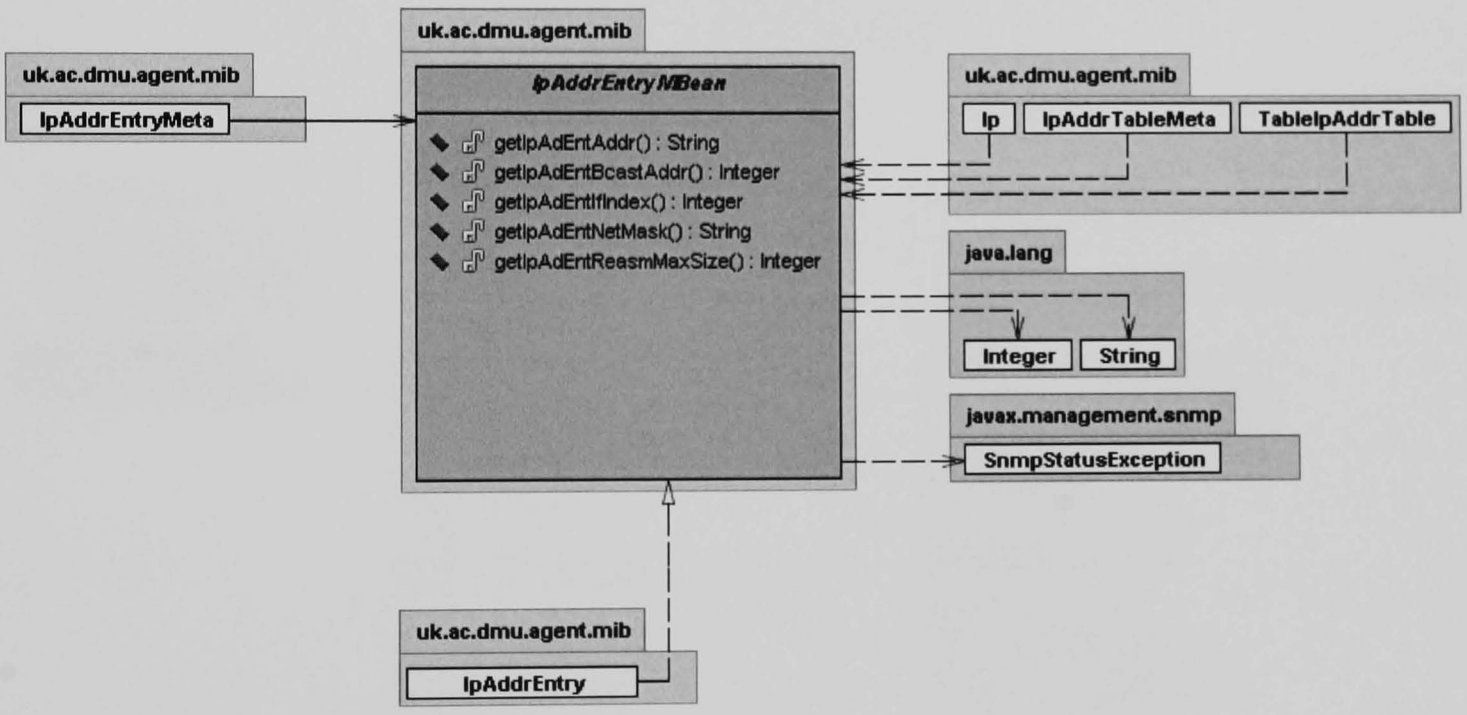
Class: Ip



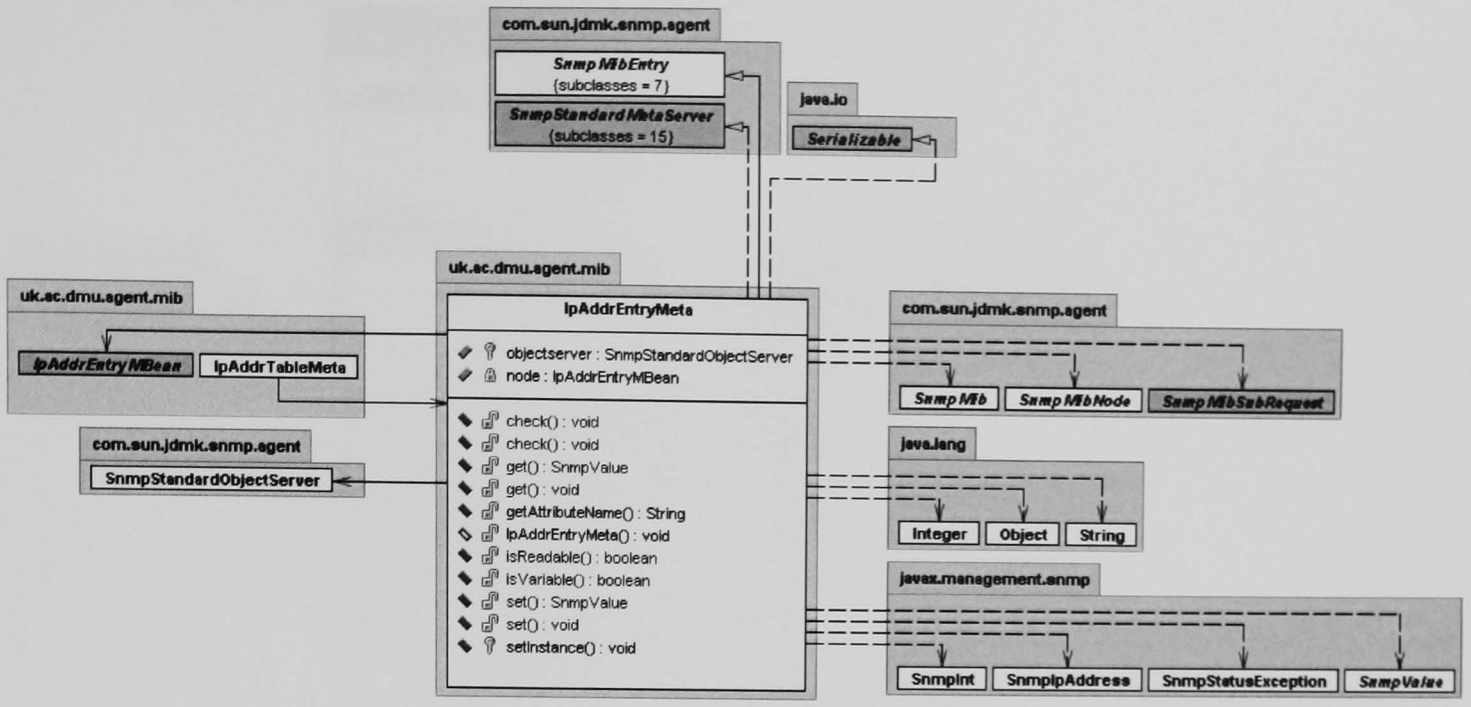
Class: IpAddrEntry



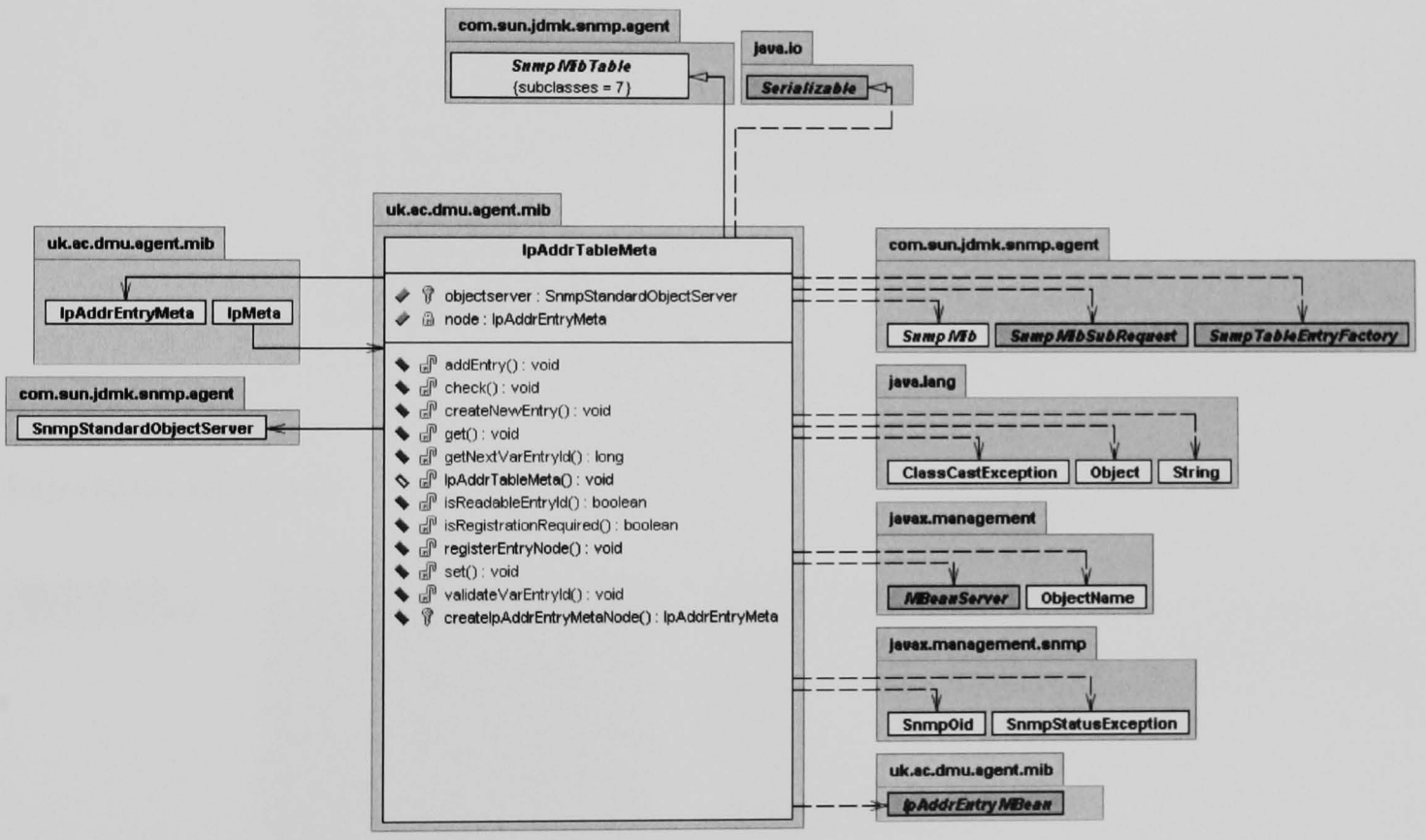
Interface: IpAddrEntryMBean



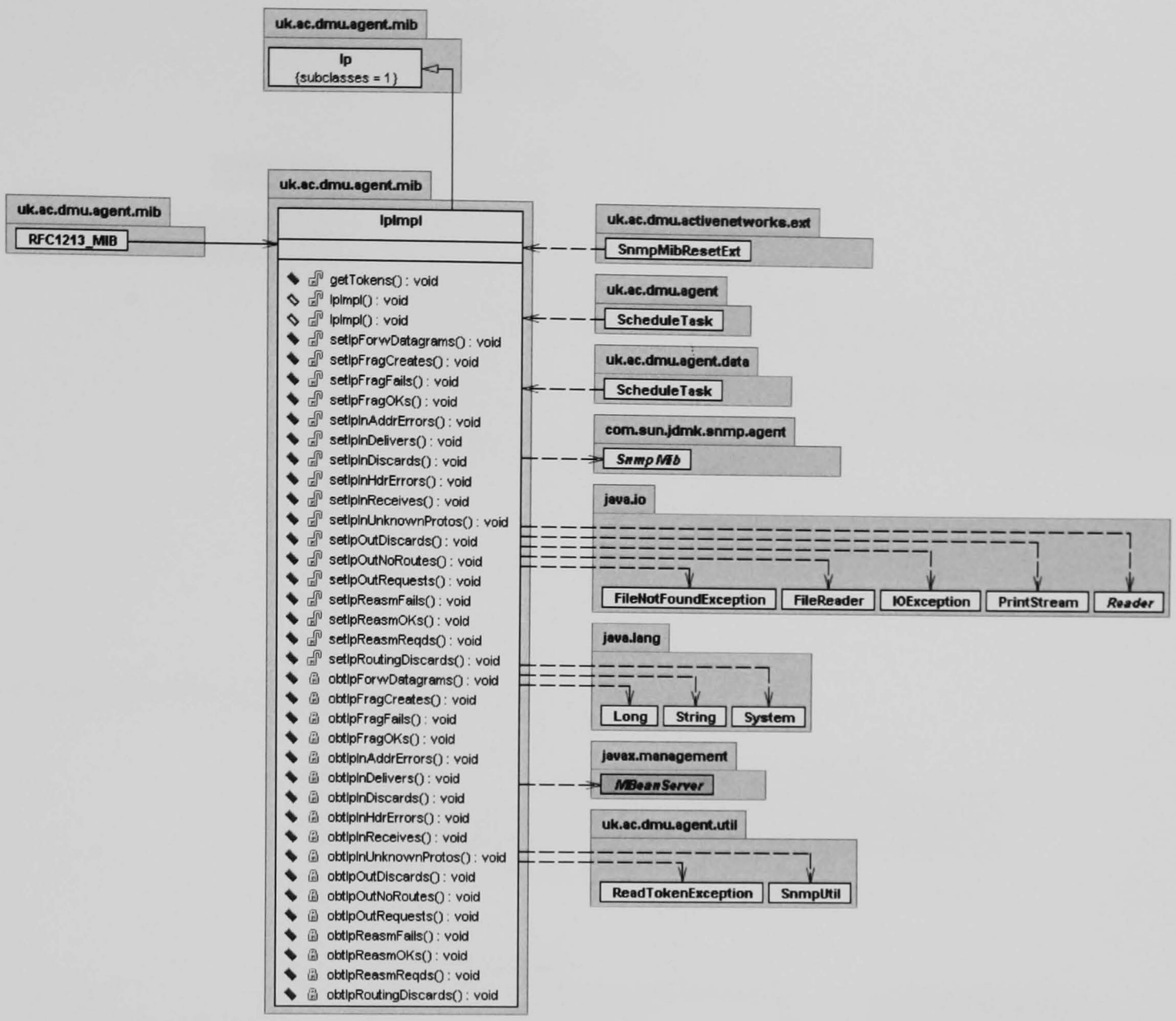
Class:IpAddrEntryMeta



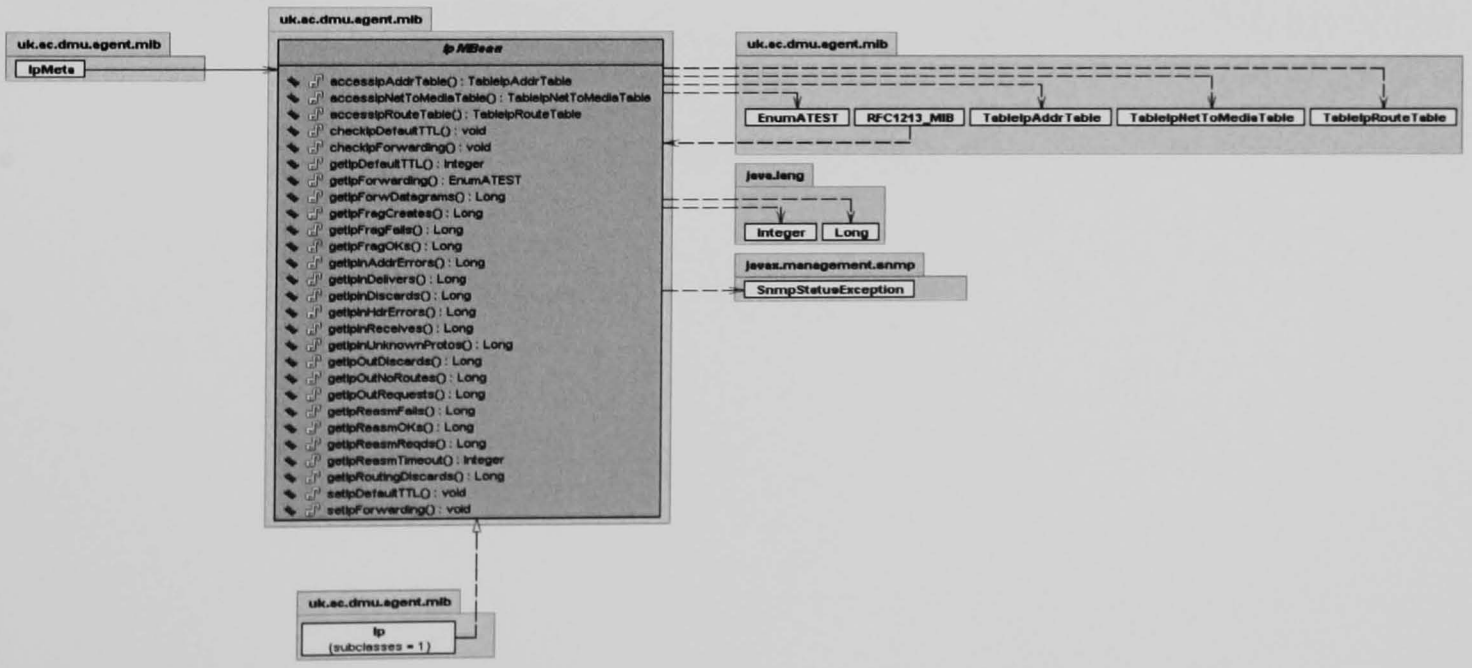
Class:IpAddrTableMeta



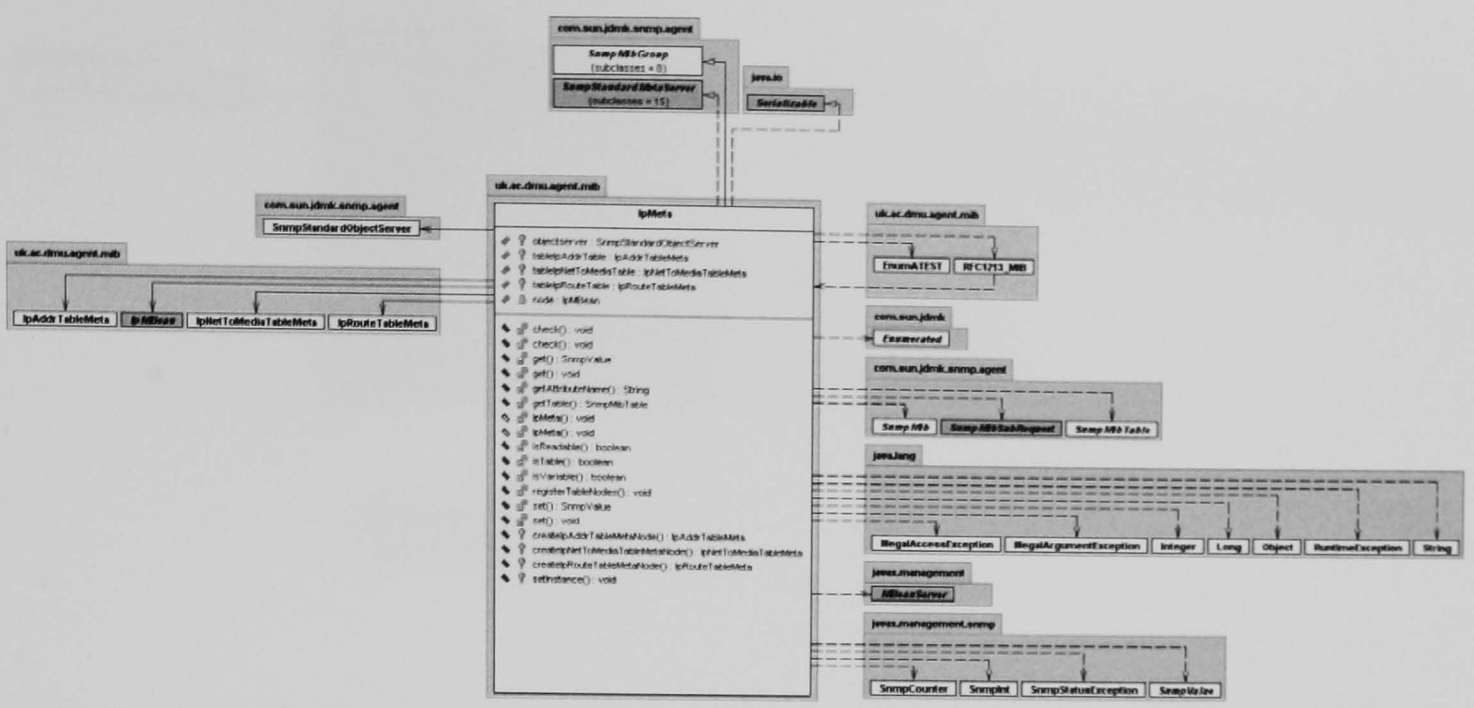
Class:IpImpl



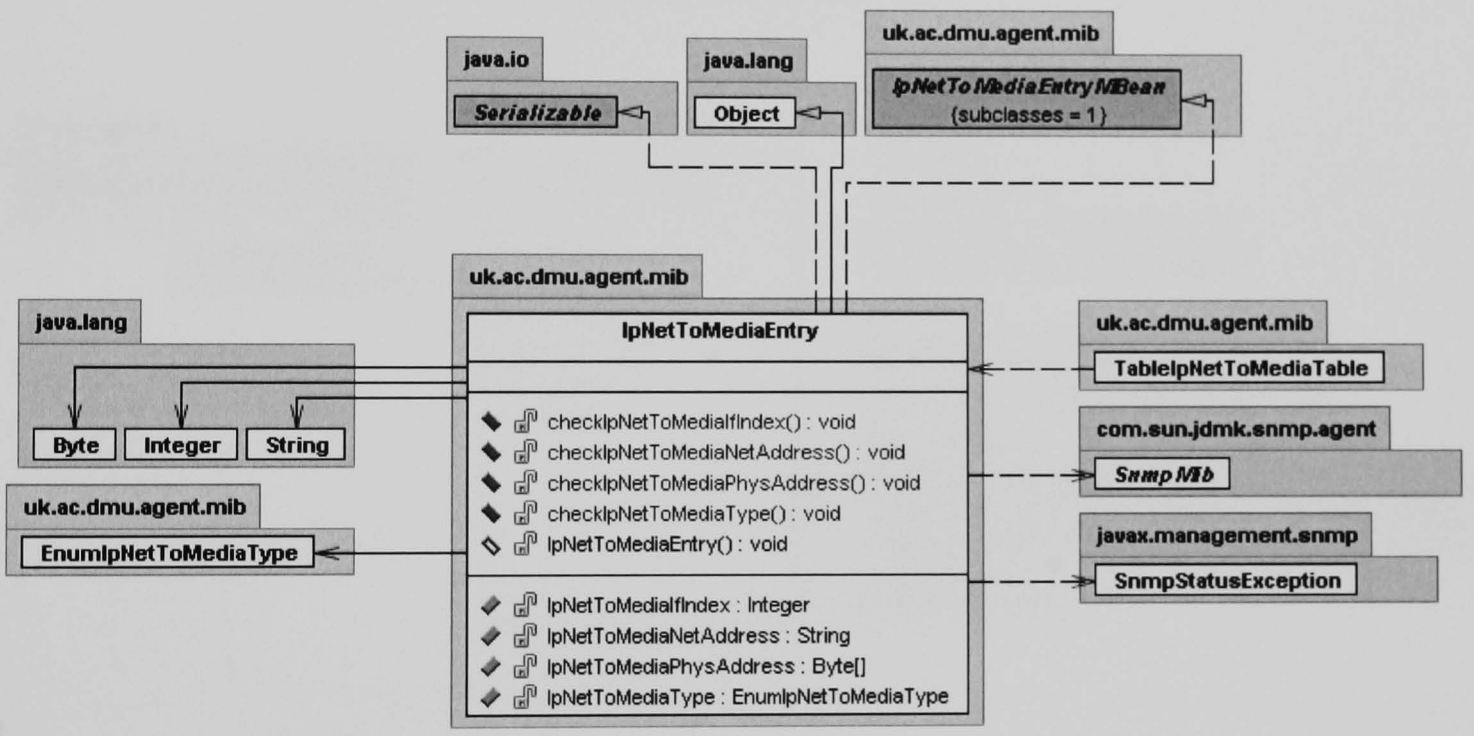
Interface: IpMbean



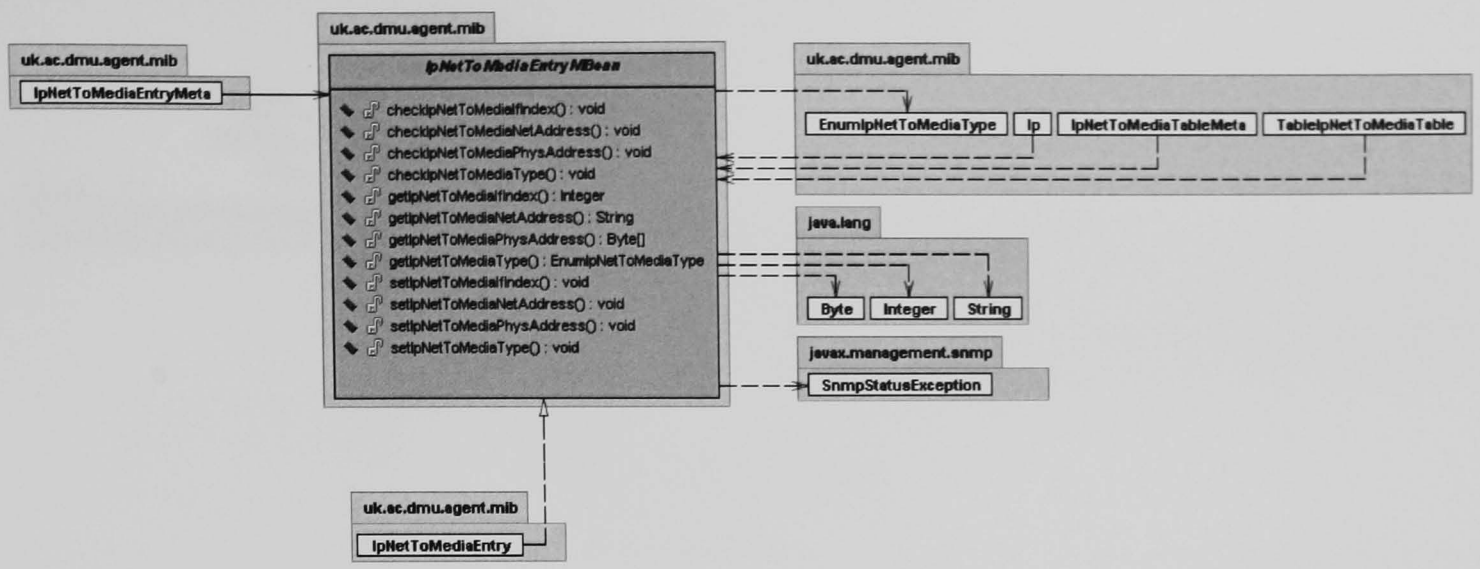
Class: IpMeta



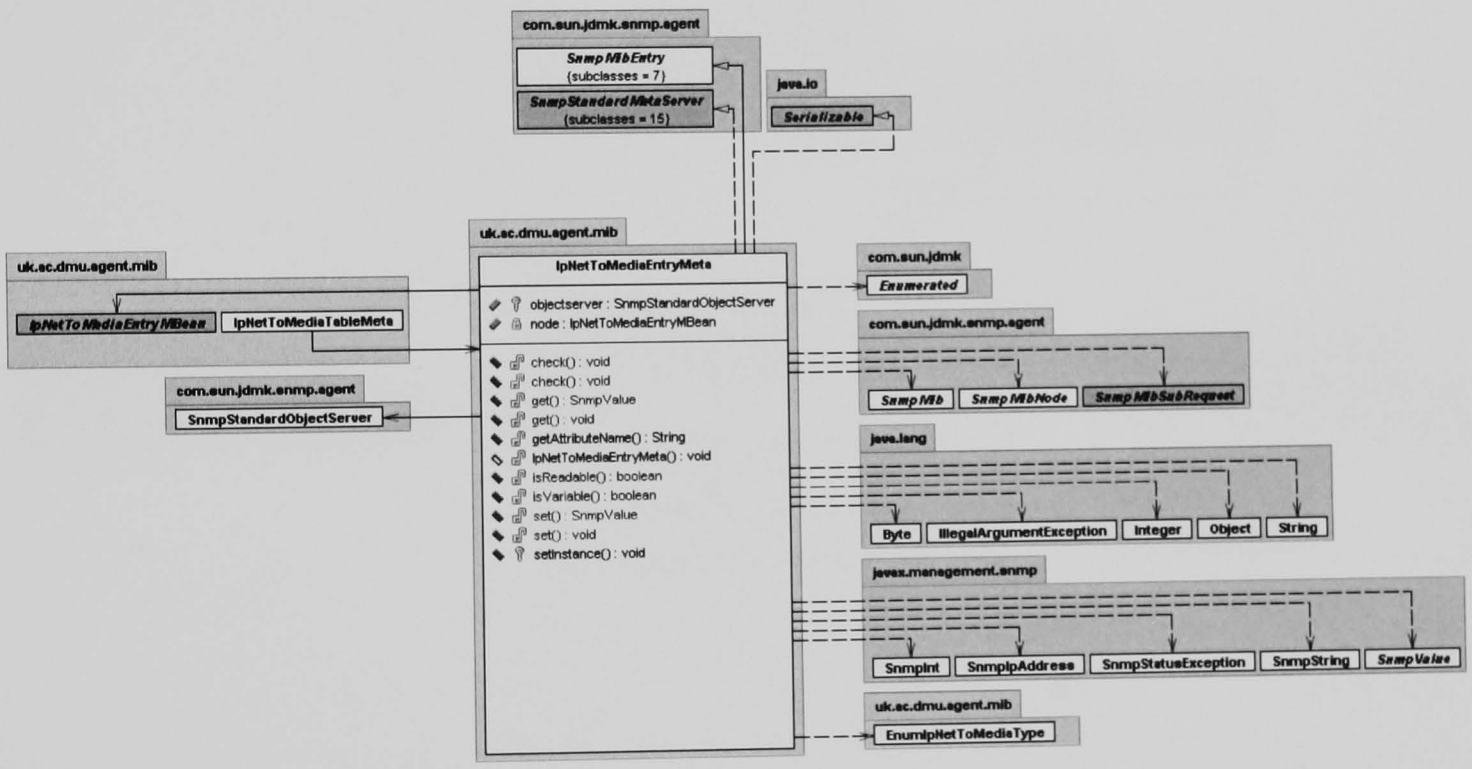
Class:IpNetToMediaEntry



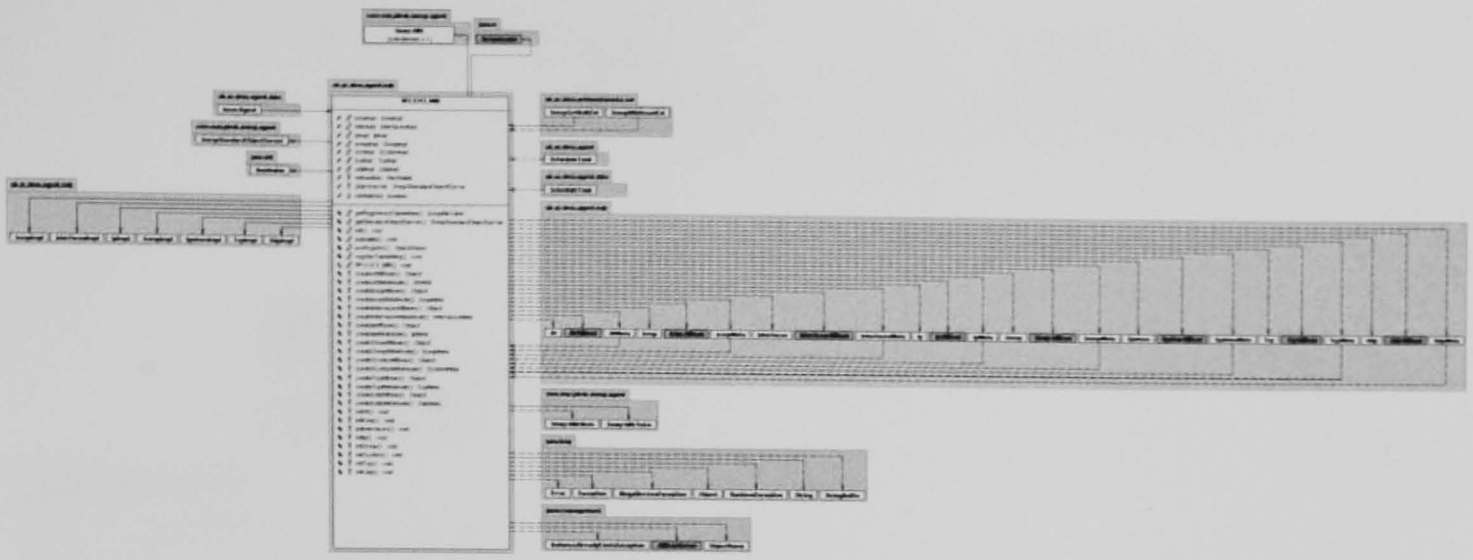
Interface:IpNetToMediaEntryMBean



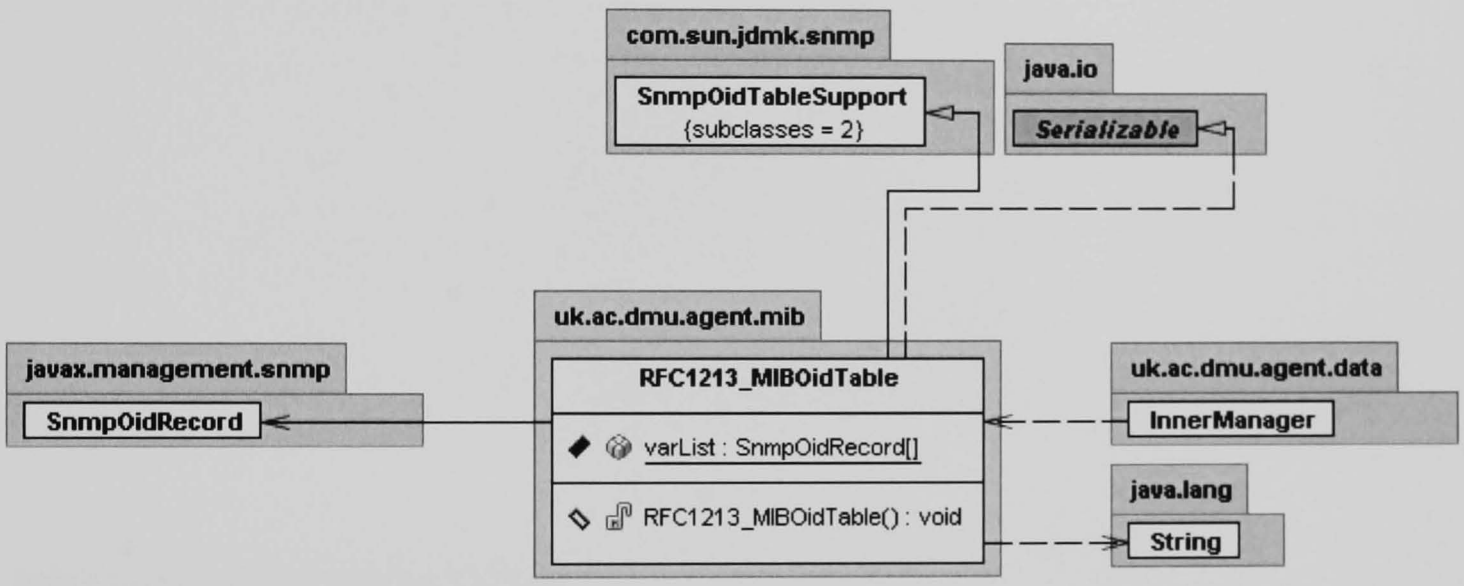
Class: IpNetToMediaEntryMeta



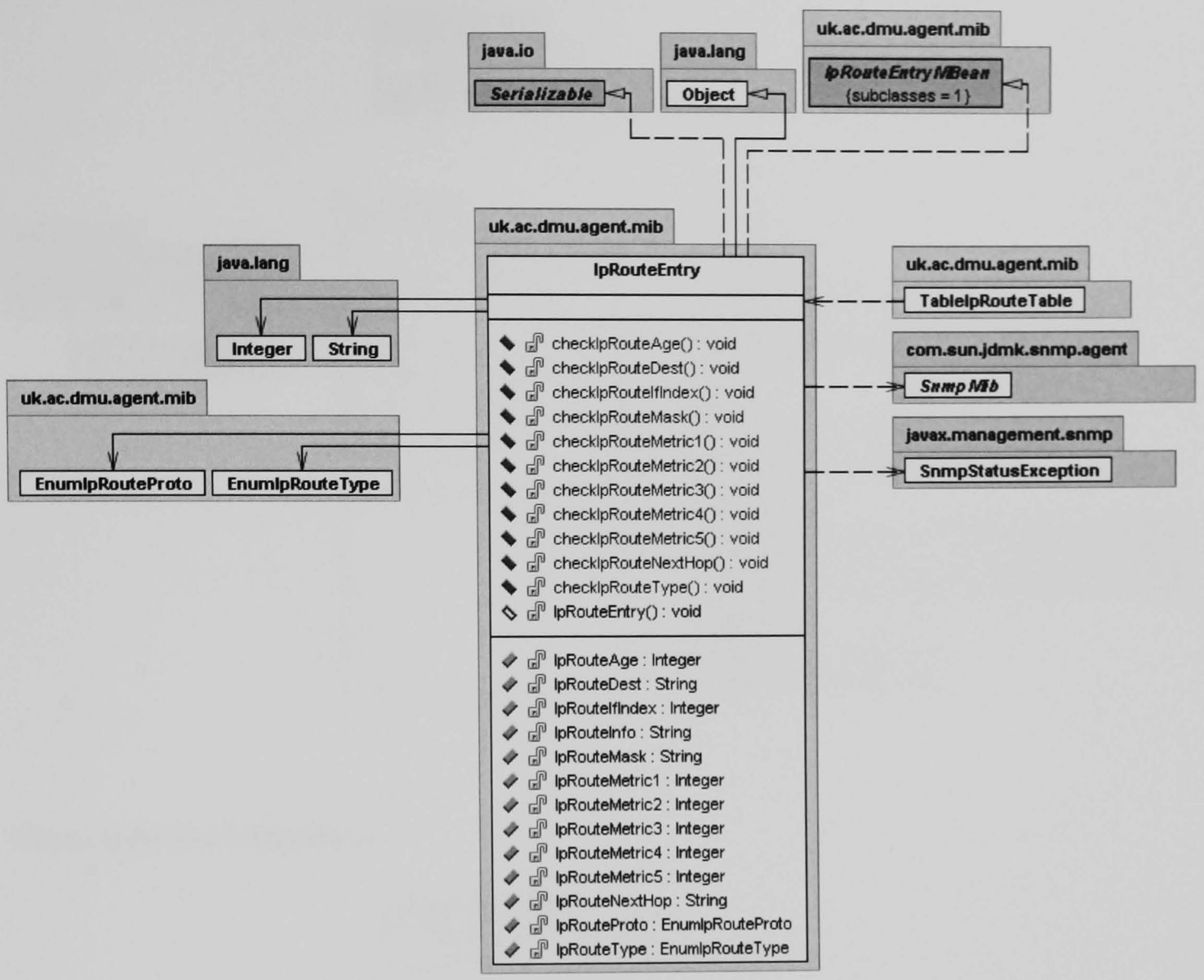
Class:RFC1213_MIB



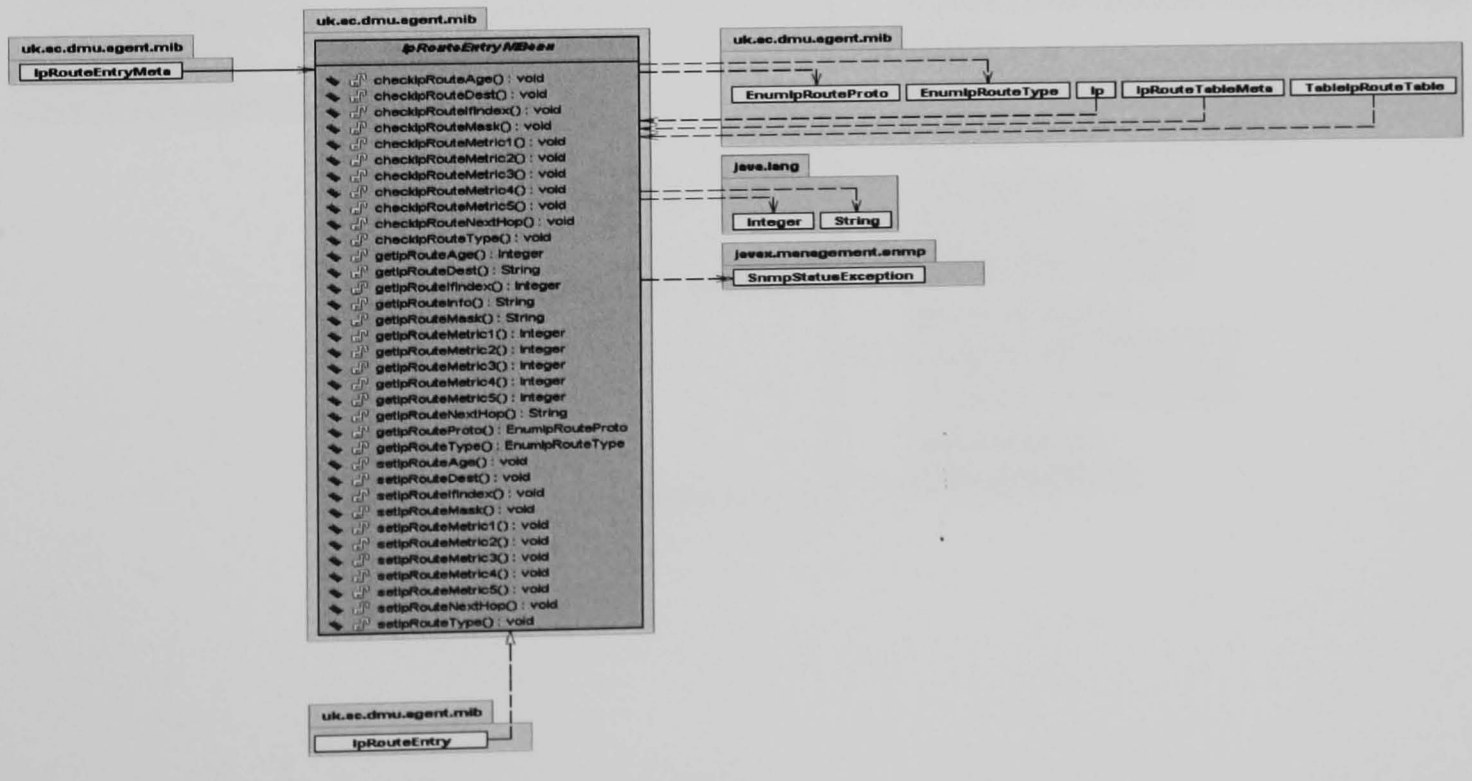
Class: RFC1213_MIBOidTable



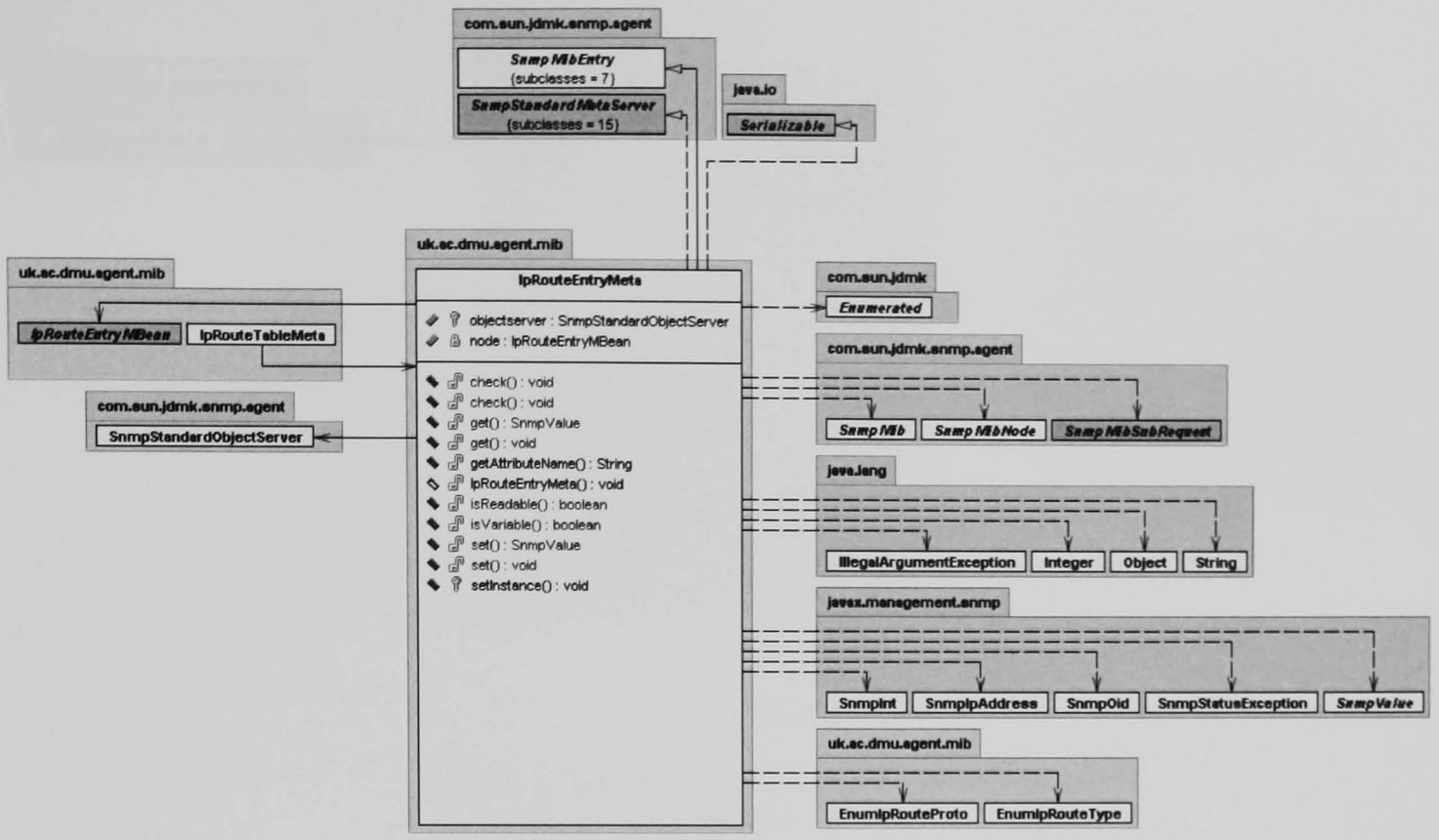
Class:IpRouteEntry



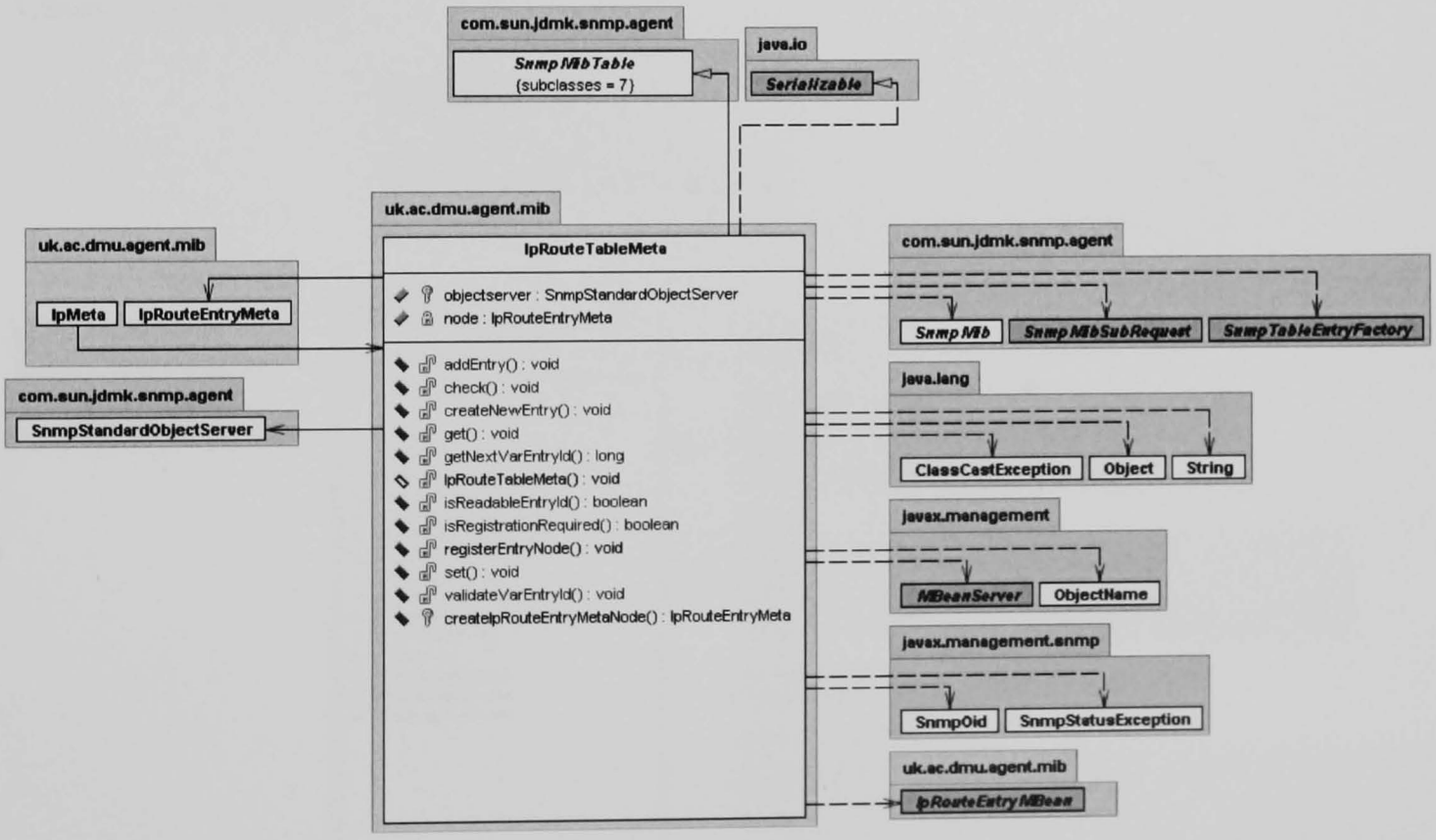
Interface: IpRouteEntryMBean



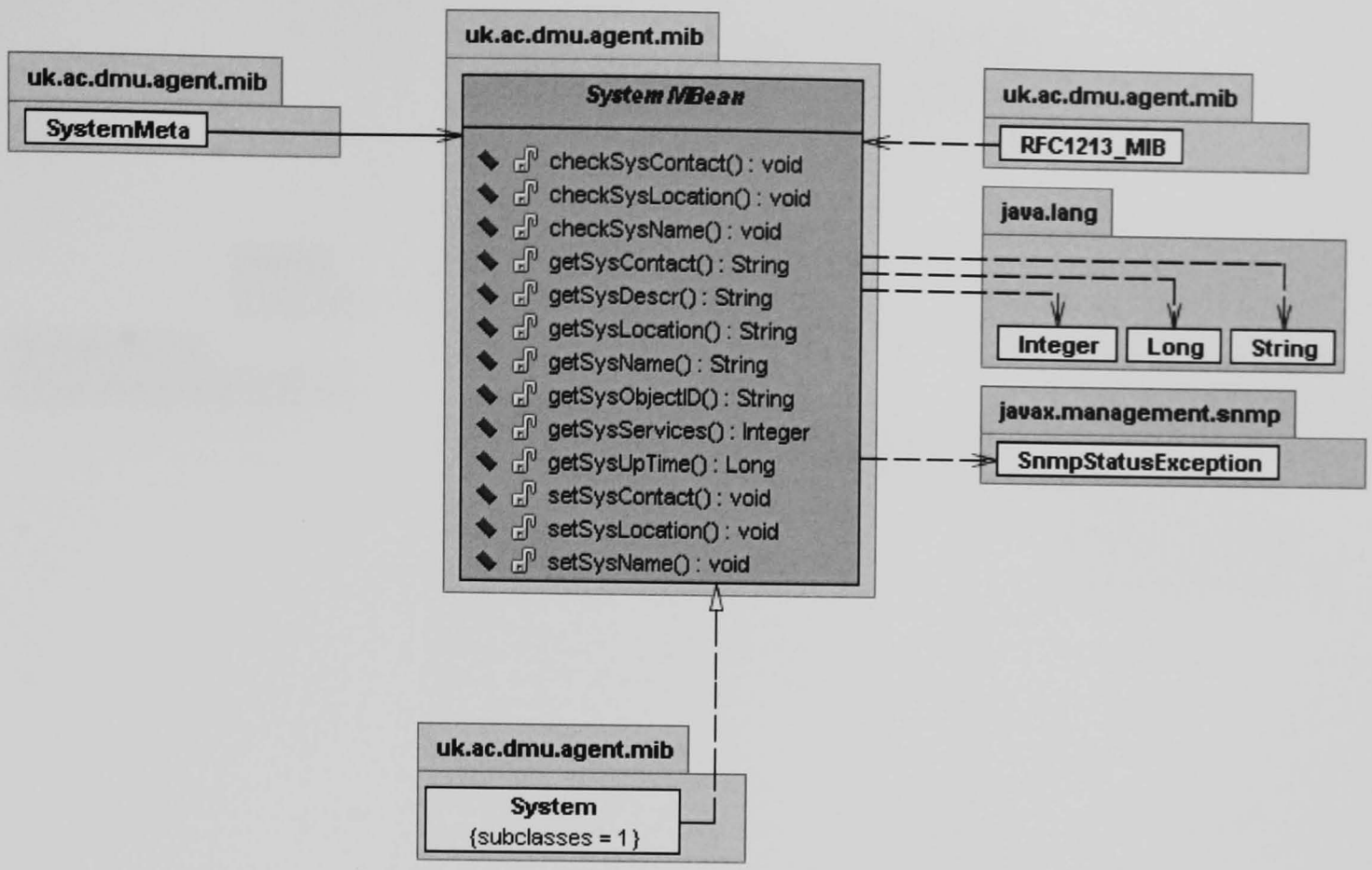
Class: IpRouteEntryMeta



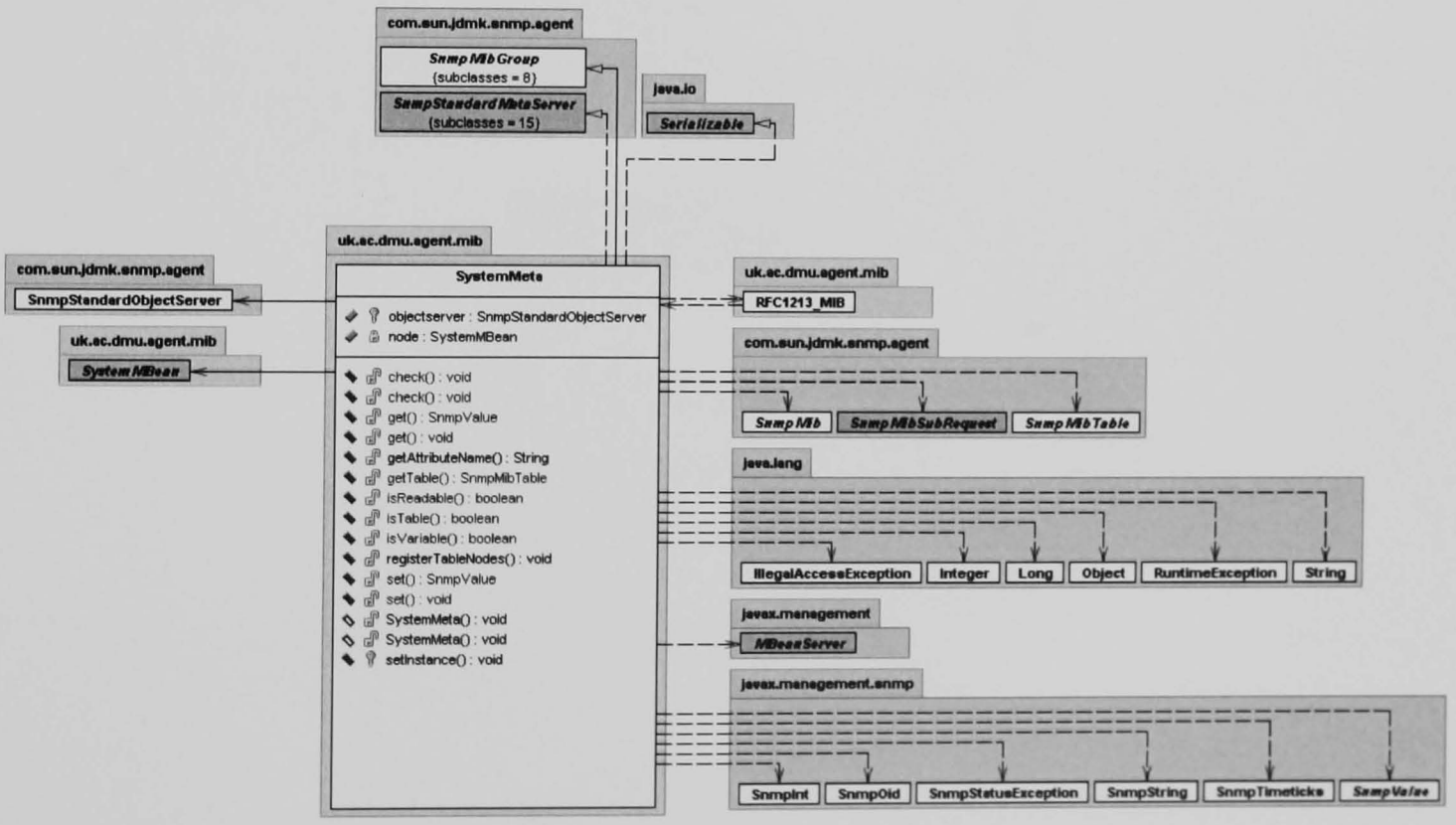
Class:IpRouteTableMeta



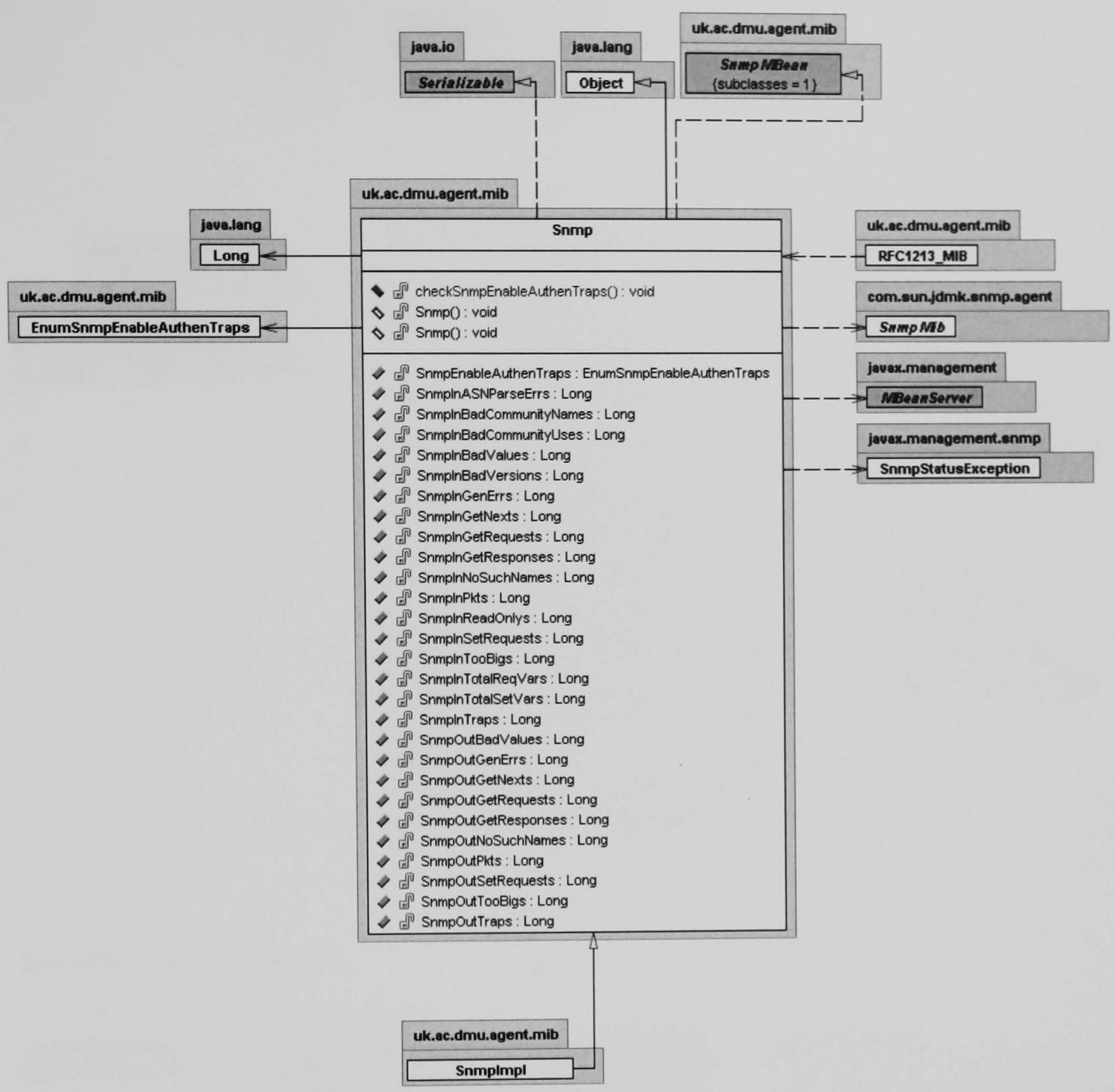
Interface: SystemMBean



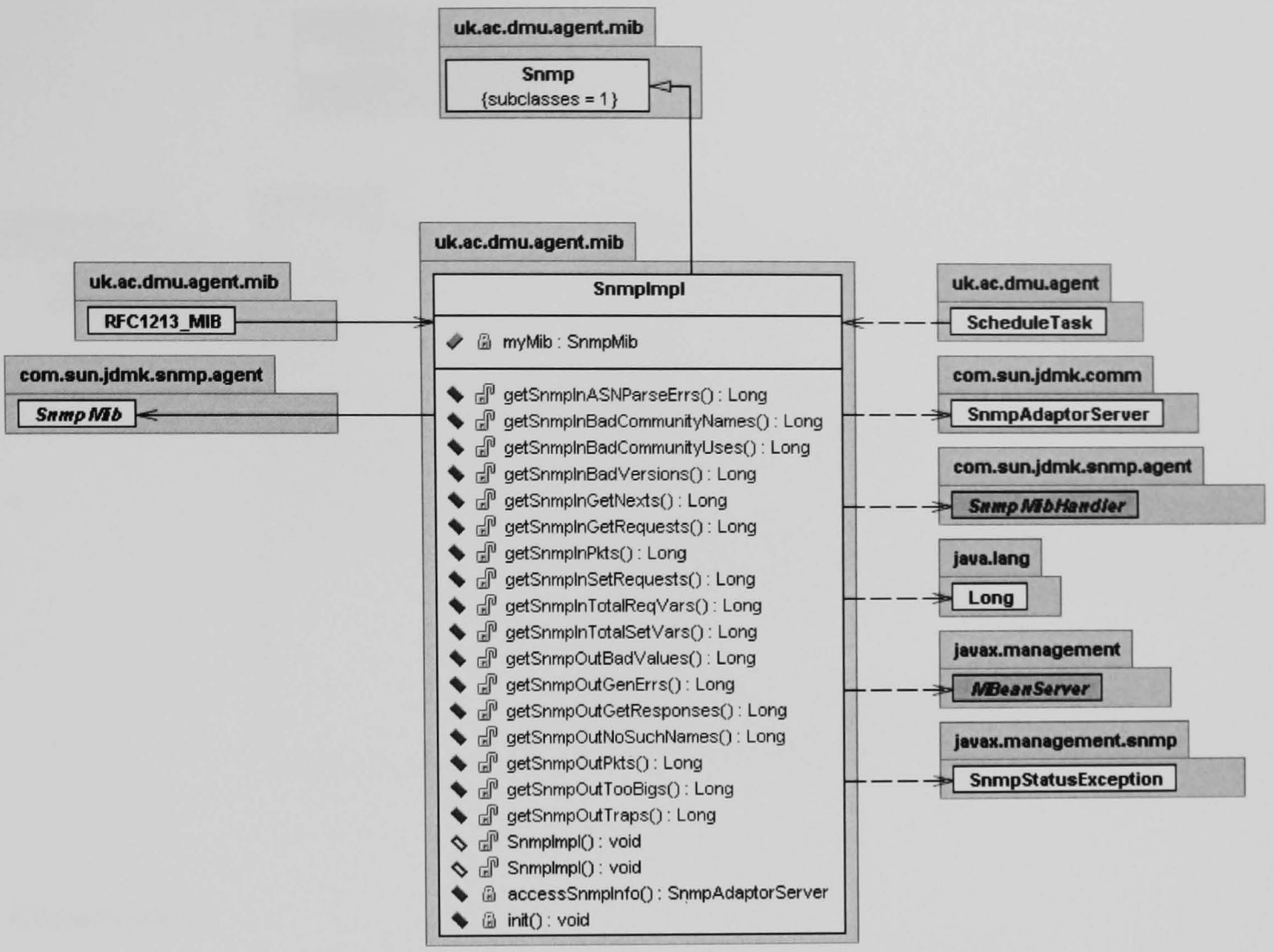
Class:SystemMeta



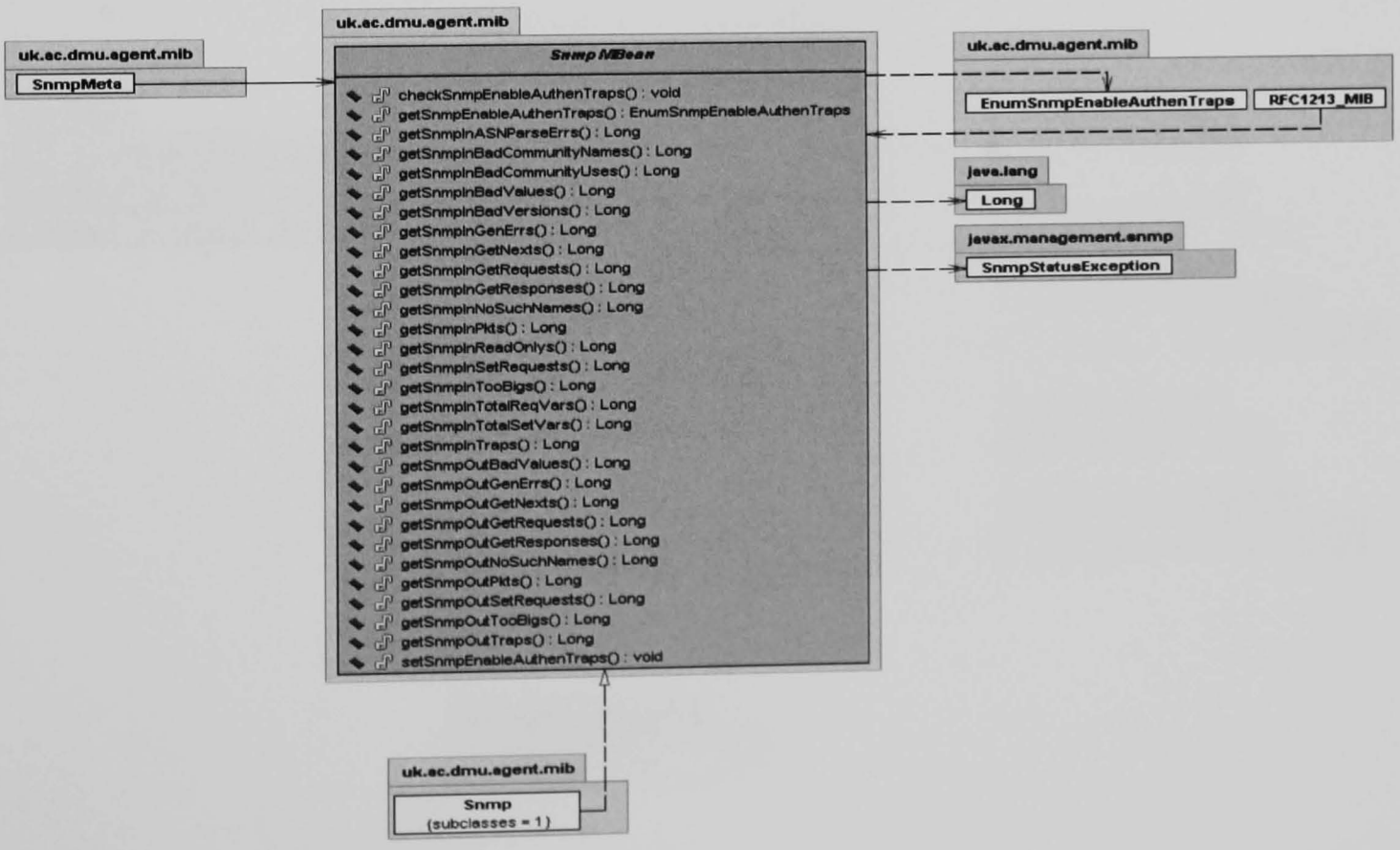
Class: Snmp


java.io
Serializable
java.lang
Object
uk.ac.dmu.agent.mib
SnmpMBean
(subclasses = 1)
uk.ac.dmu.agent.mib
Snmp
java.lang
Long
uk.ac.dmu.agent.mib
EnumSnmpEnableAuthenTraps
checkSnmpEnableAuthenTraps() : void
Snmp() : void
Snmp() : void
SnmpEnableAuthenTraps : EnumSnmpEnableAuthenTraps
SnmpInASNParseErrs : Long
SnmpInBadCommunityNames : Long
SnmpInBadCommunityUses : Long
SnmpInBadValues : Long
SnmpInBadVersions : Long
SnmpInGenErrs : Long
SnmpInGetNexts : Long
SnmpInGetRequests : Long
SnmpInGetResponses : Long
SnmpInNoSuchNames : Long
SnmpInPkts : Long
SnmpInReadOnlys : Long
SnmpInSetRequests : Long
SnmpInTooBigs : Long
SnmpInTotalReqVars : Long
SnmpInTotalSetVars : Long
SnmpInTraps : Long
SnmpOutBadValues : Long
SnmpOutGenErrs : Long
SnmpOutGetNexts : Long
SnmpOutGetRequests : Long
SnmpOutGetResponses : Long
SnmpOutNoSuchNames : Long
SnmpOutPkts : Long
SnmpOutSetRequests : Long
SnmpOutTooBigs : Long
SnmpOutTraps : Long
uk.ac.dmu.agent.mib
RFC1213_MIB
com.sun.jdmk.snmp.agent
SnmpMib
javax.management
MBeanServer
javax.management.snmp
SnmpStatusException
uk.ac.dmu.agent.mib
SnmpImpl

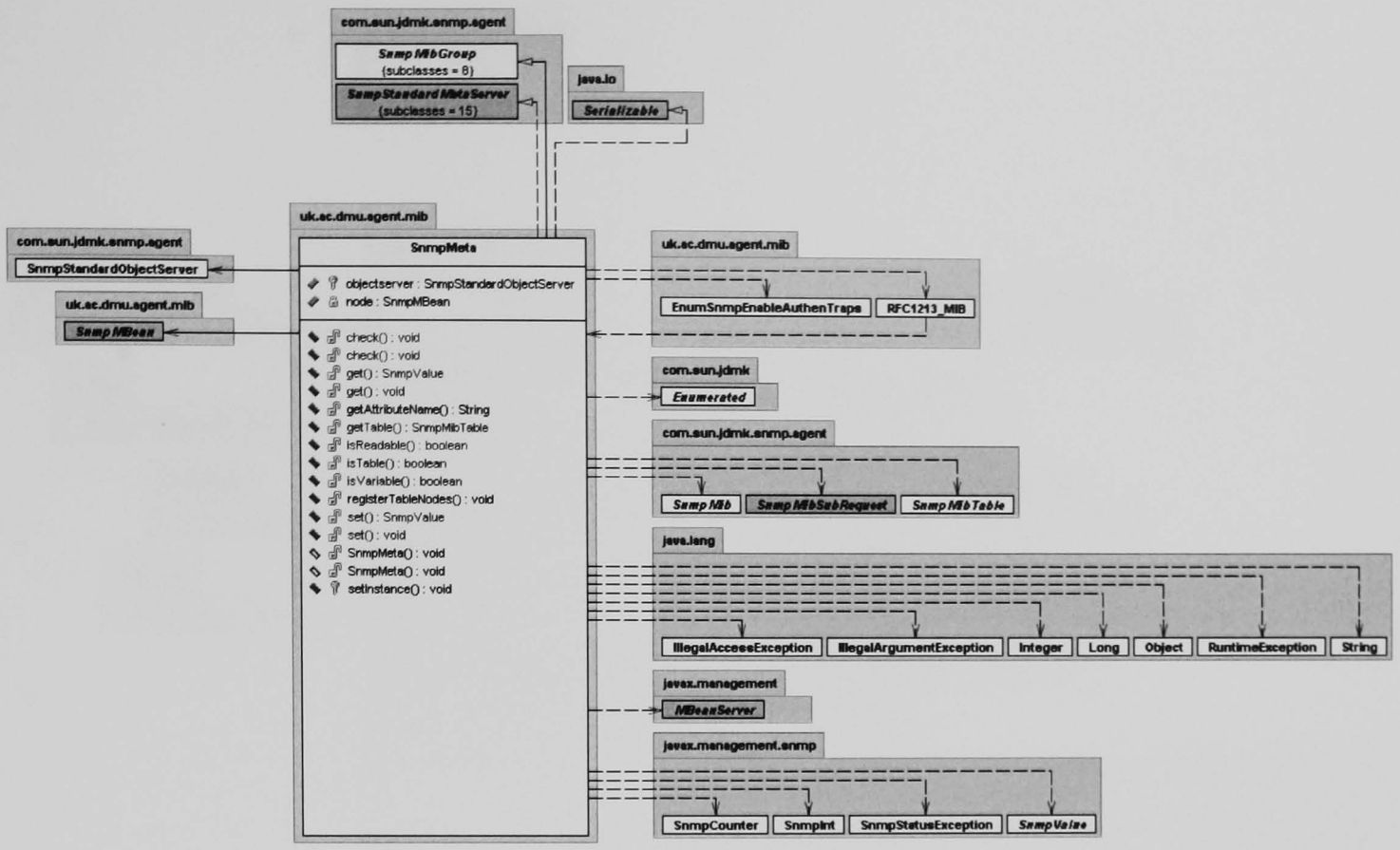
Class:SnmpImpl



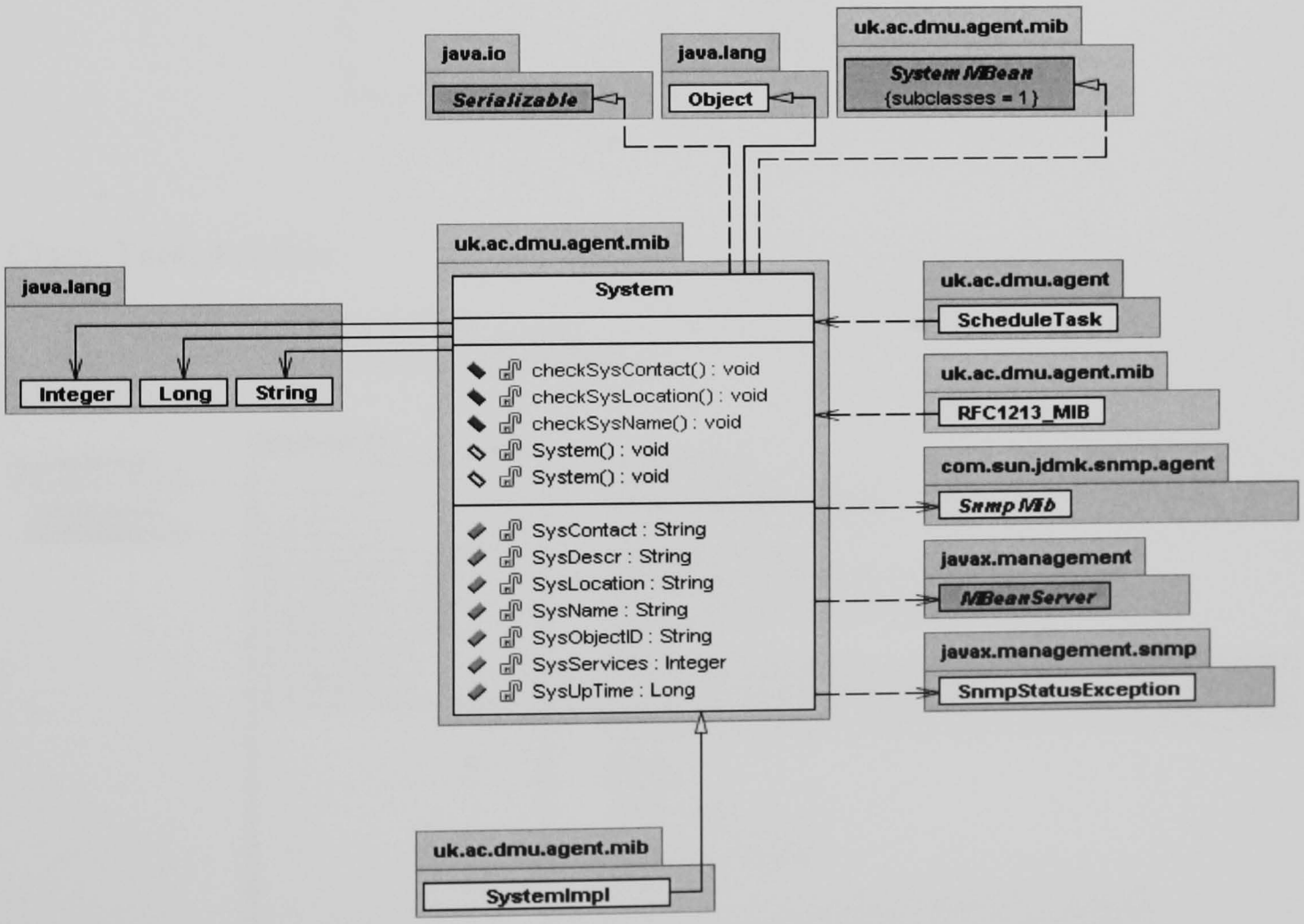
Interface: SnmpMBean



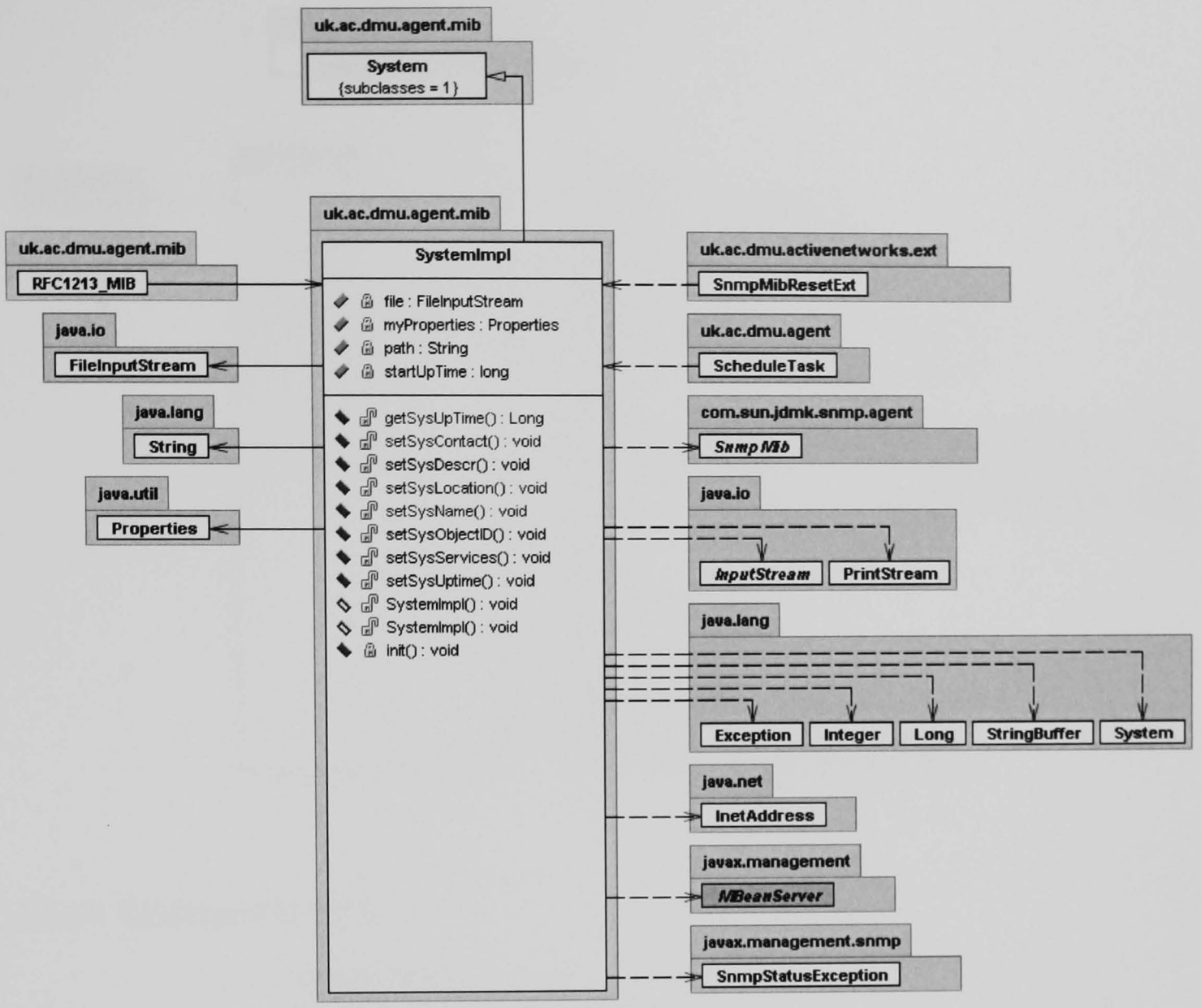
Class:SnmpMeta



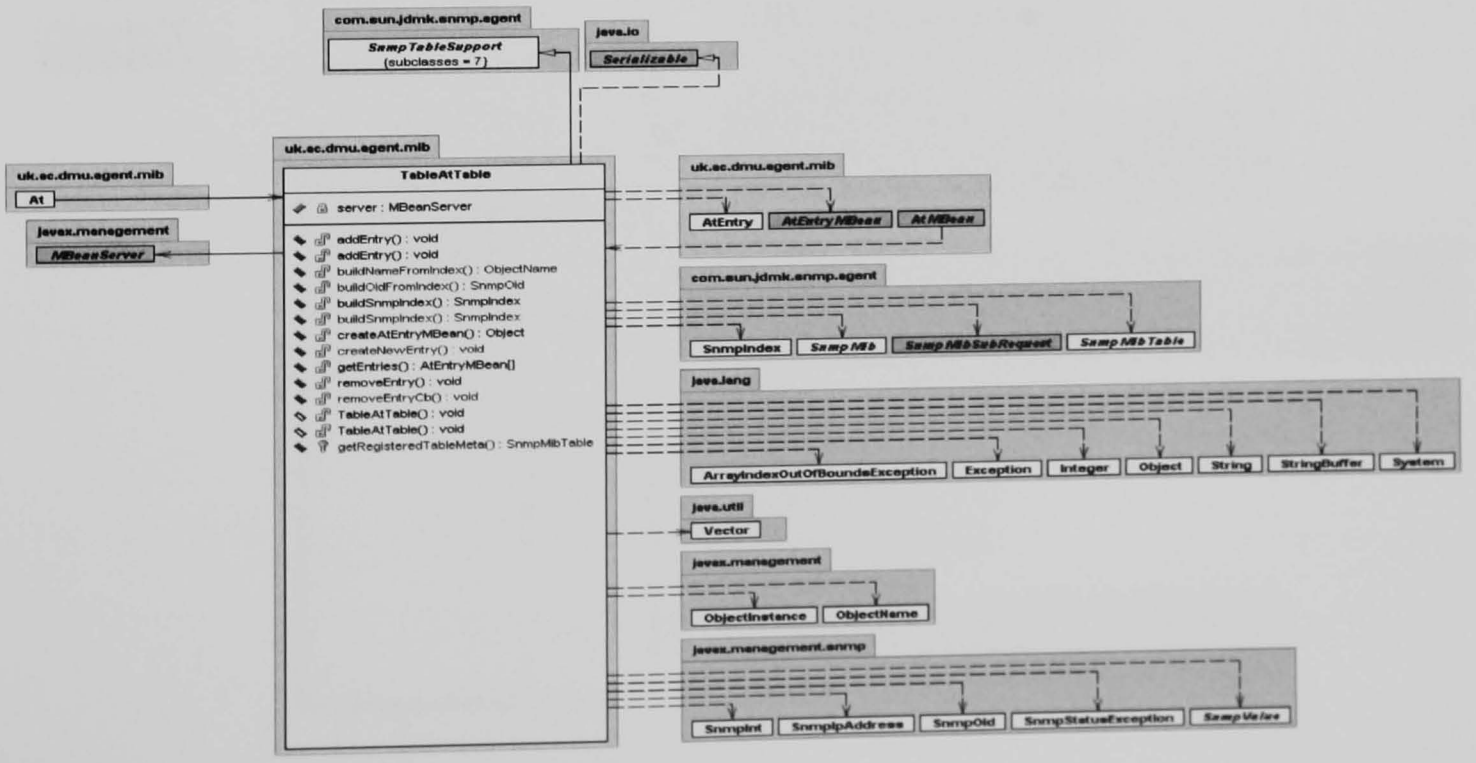
Class:System



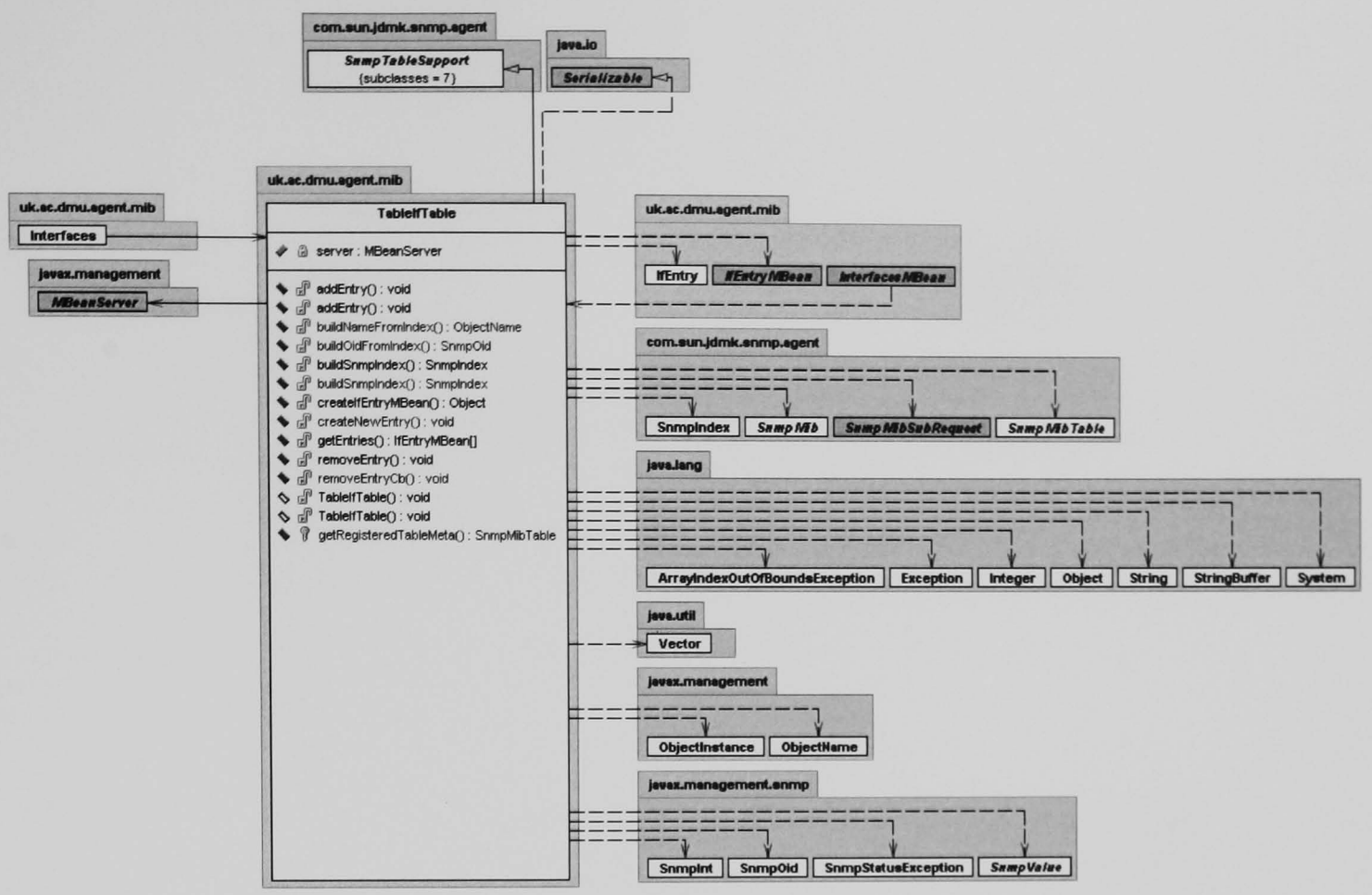
Class:SystemImpl



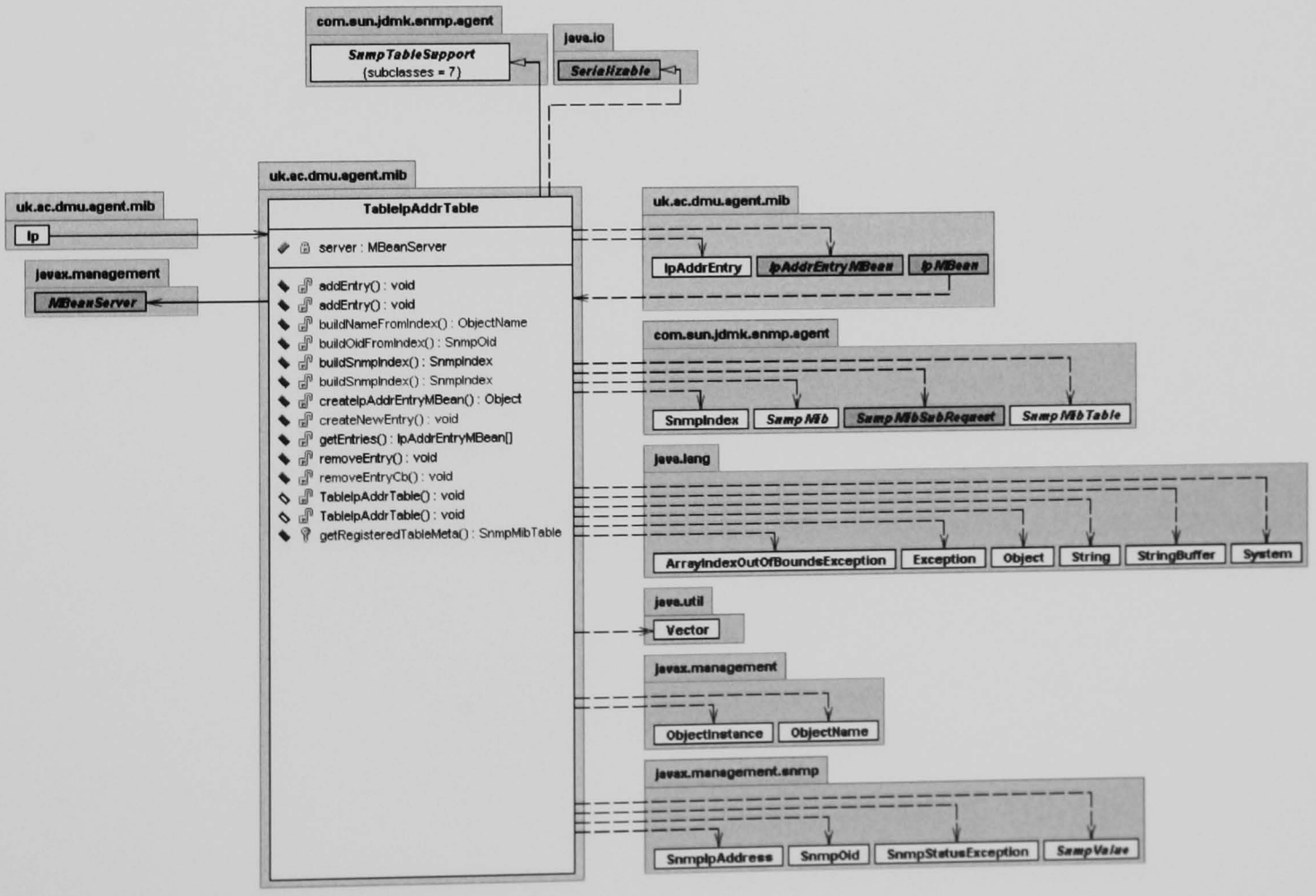
Class: TableAtTable



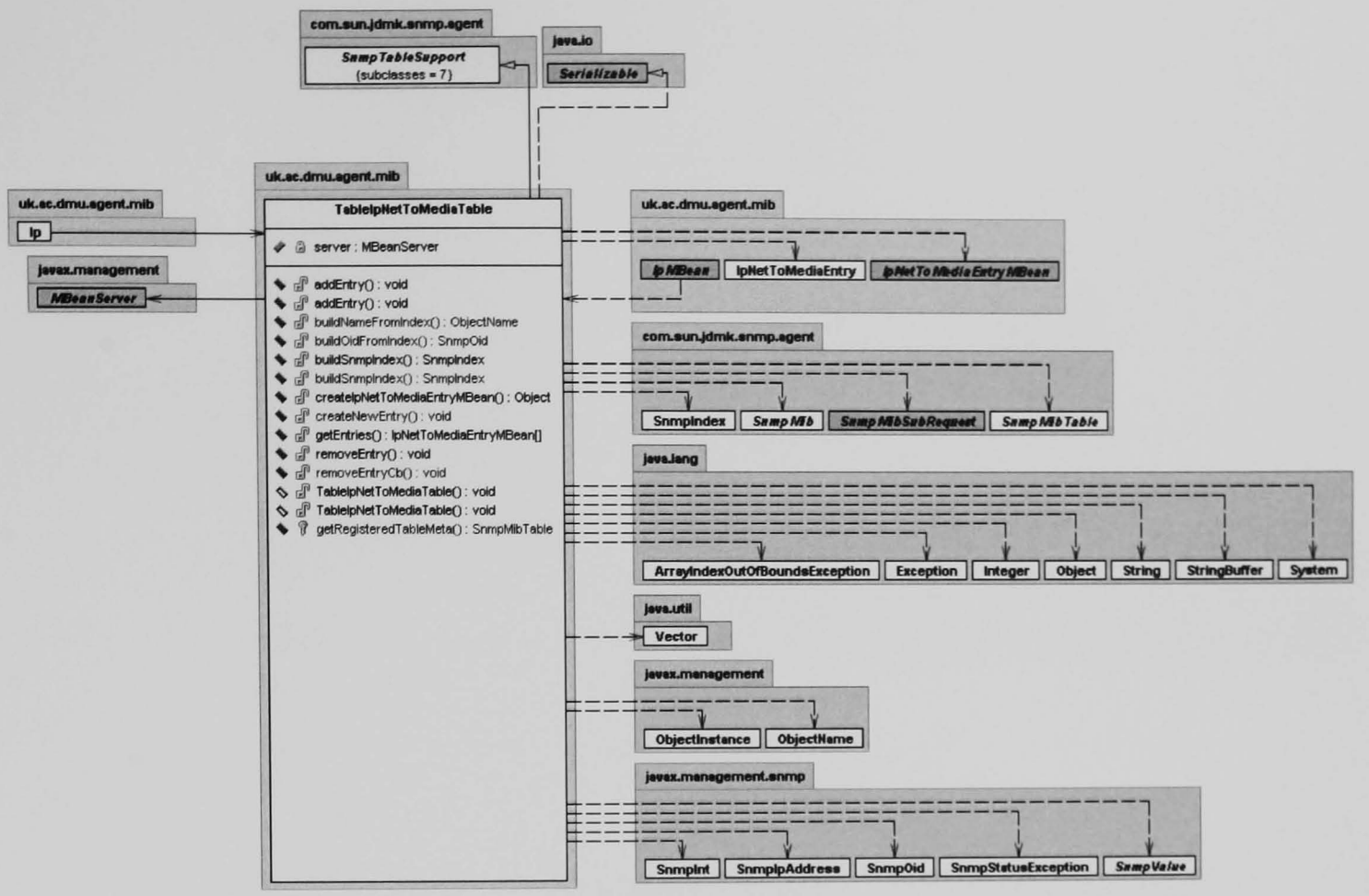
Class: TableIfTable



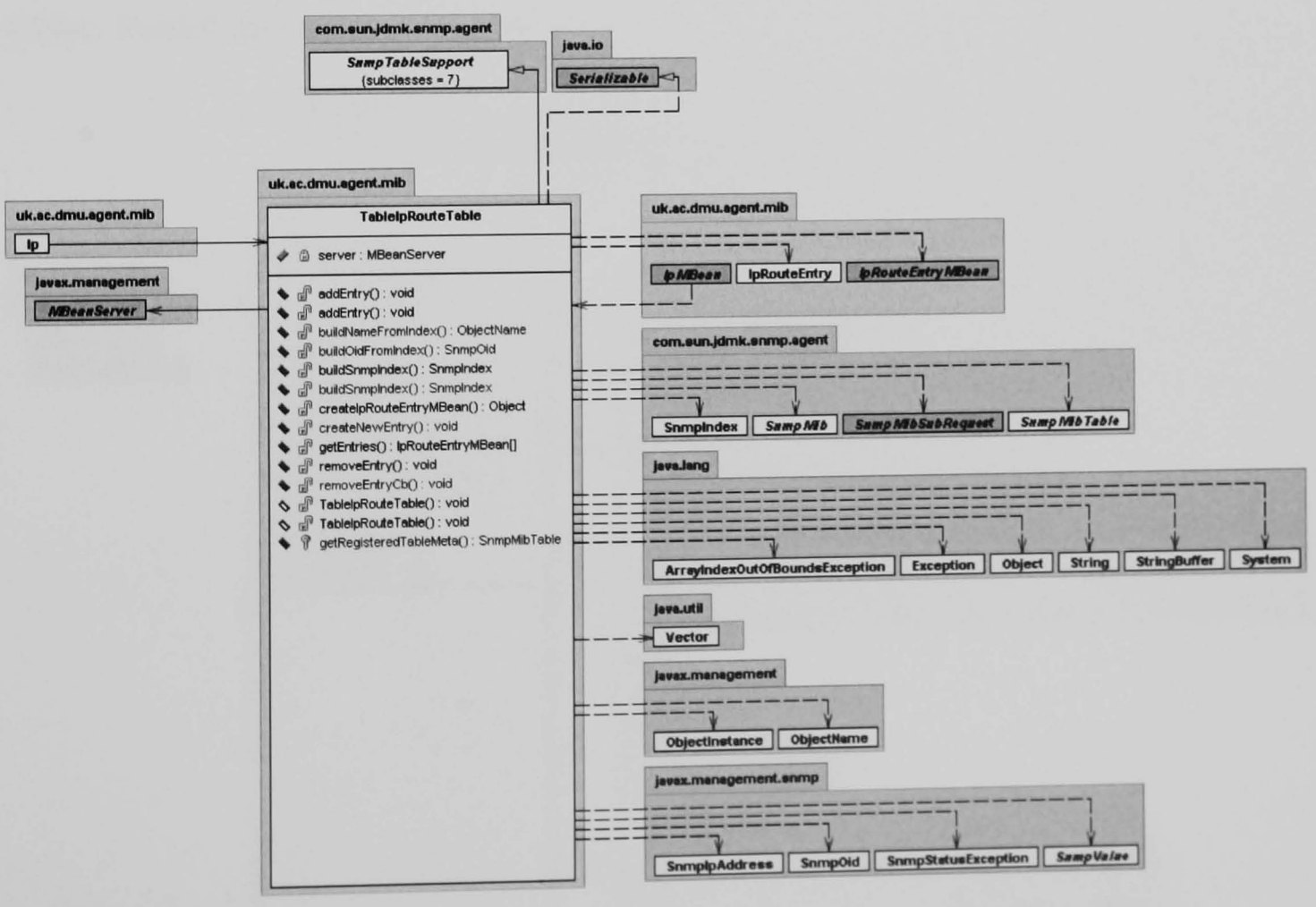
Class: TableIpAddrTable



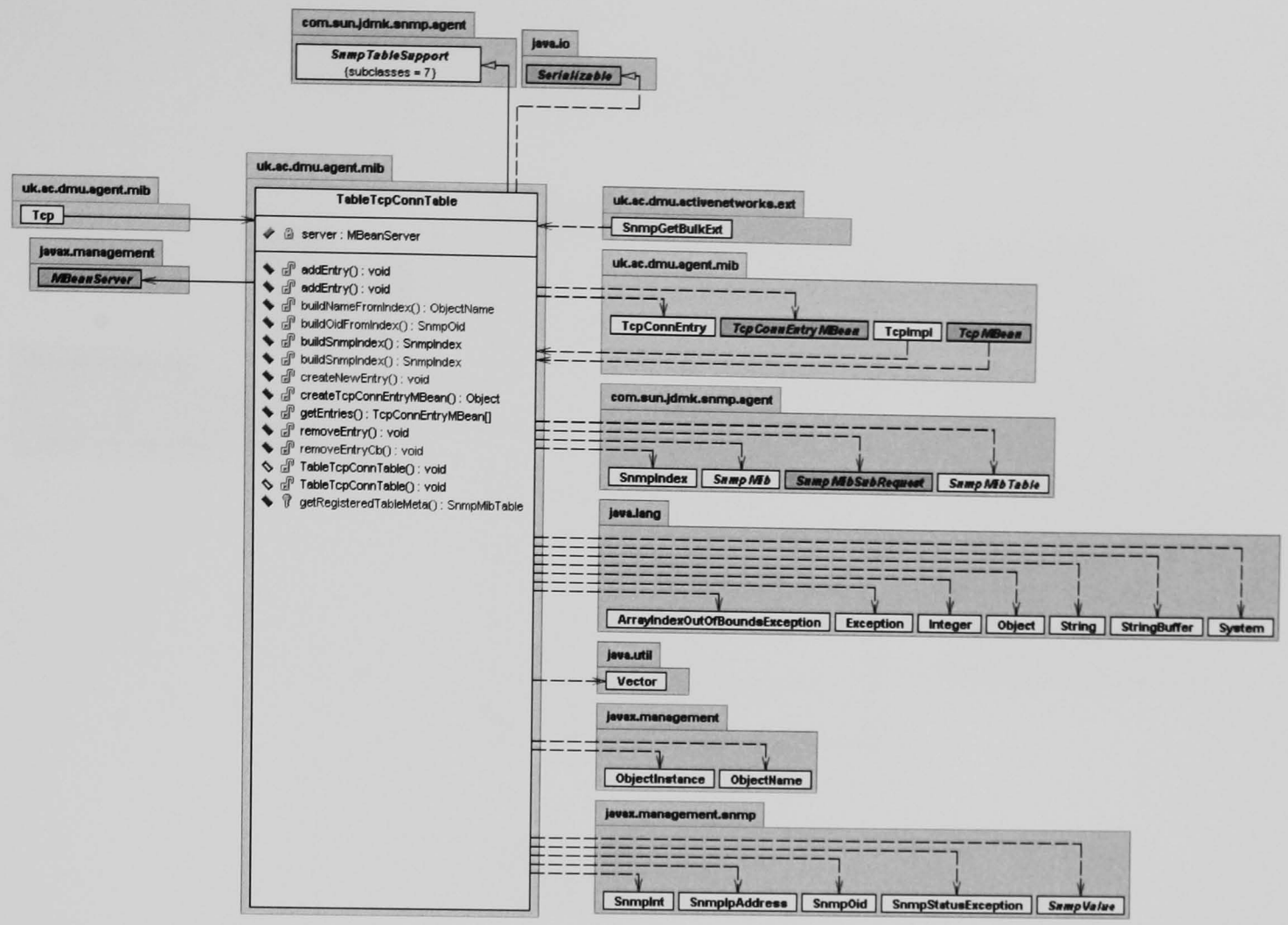
Class: TableIpNetToMediaTable



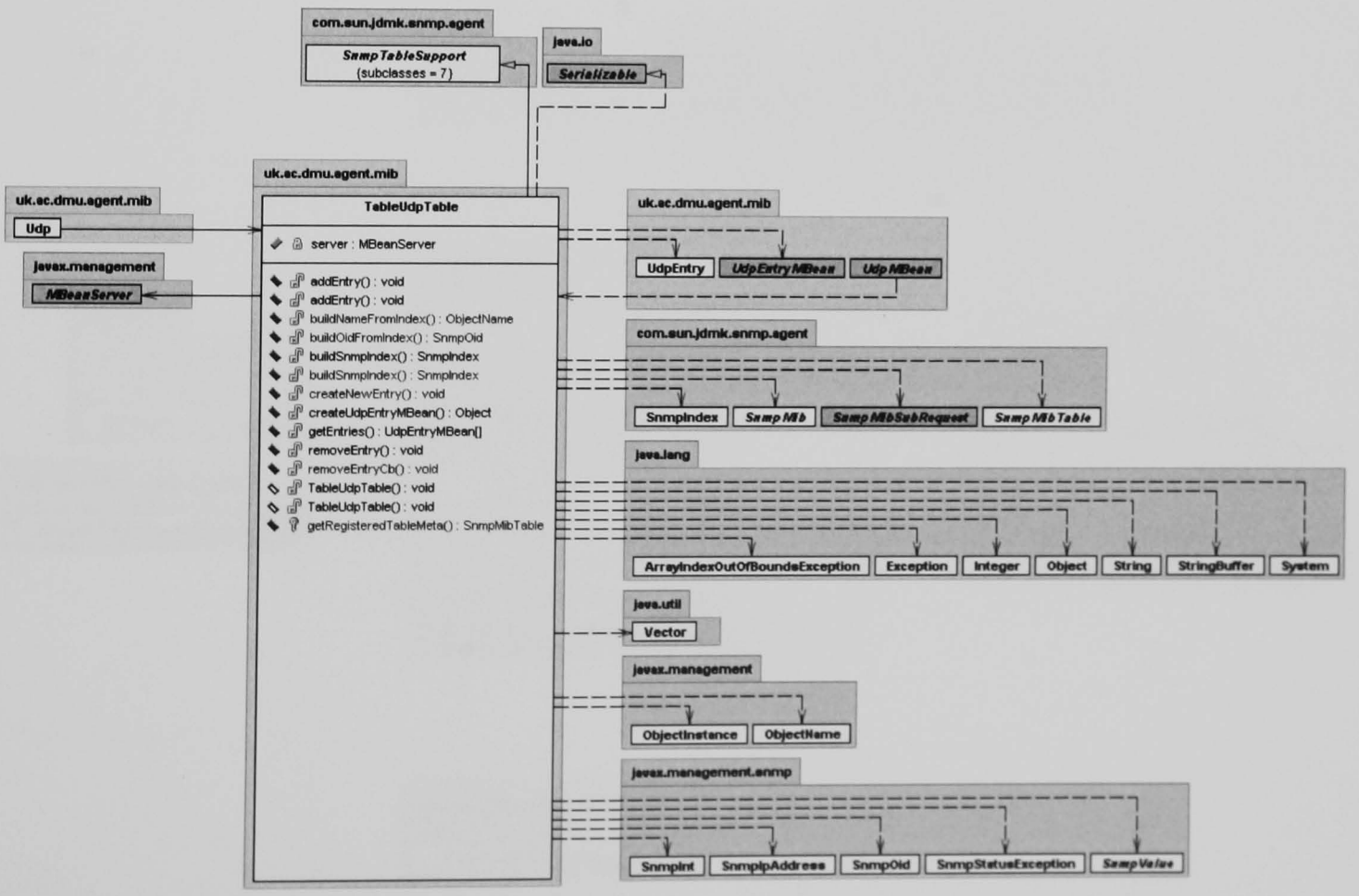
Class: TableIpRouteTable



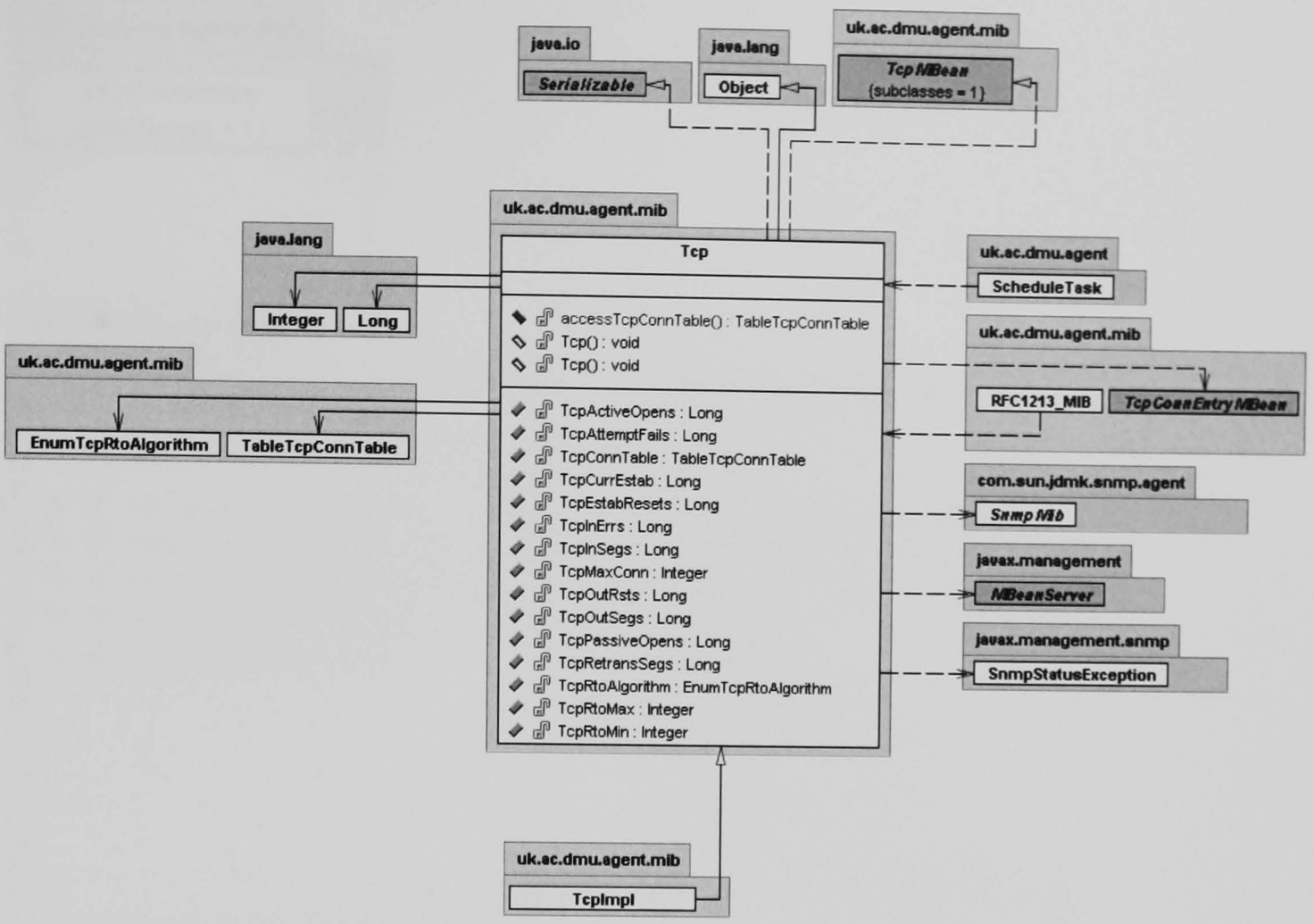
Class: TableTcpConnTable



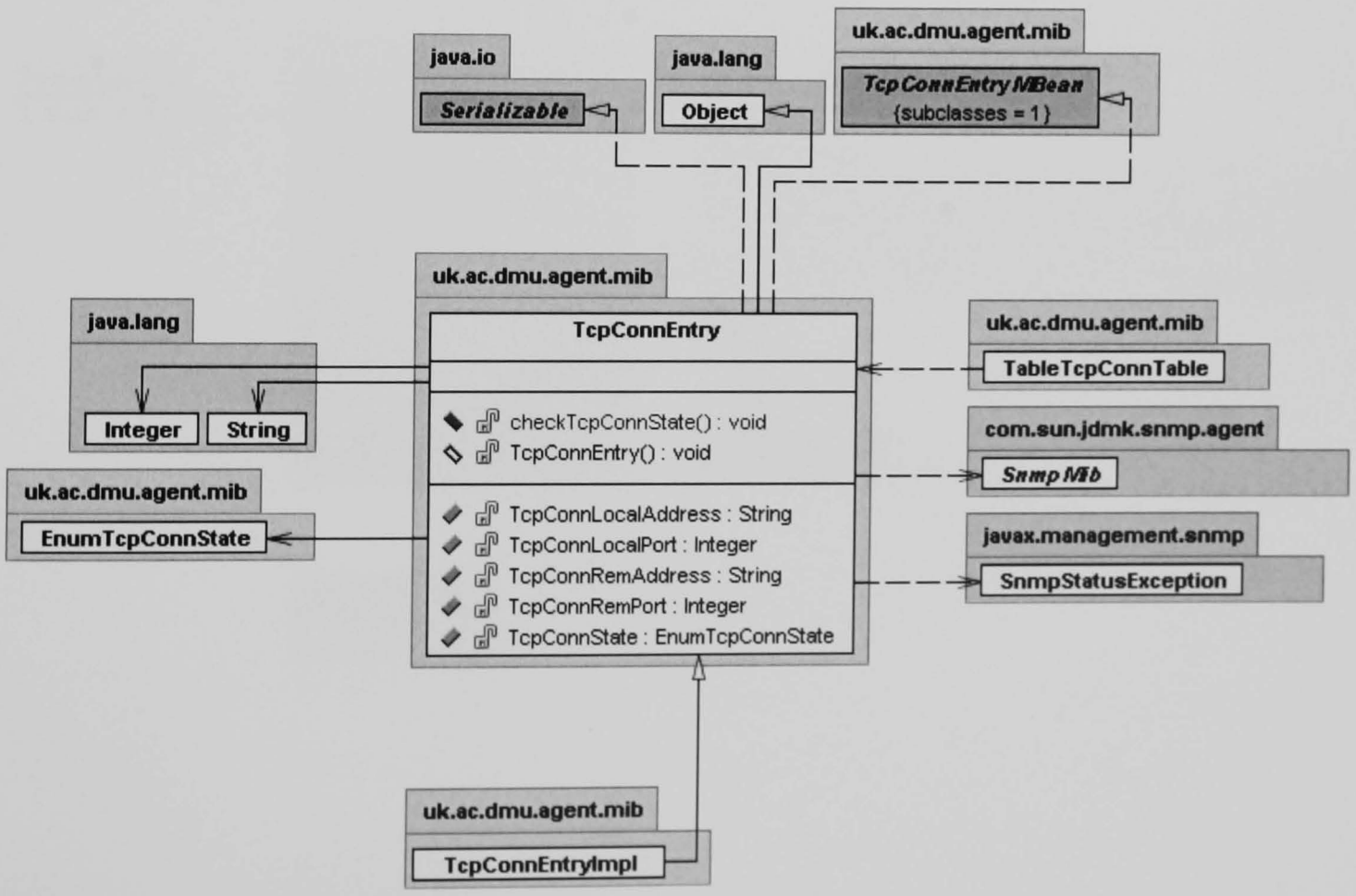
Class: TableUdpTable



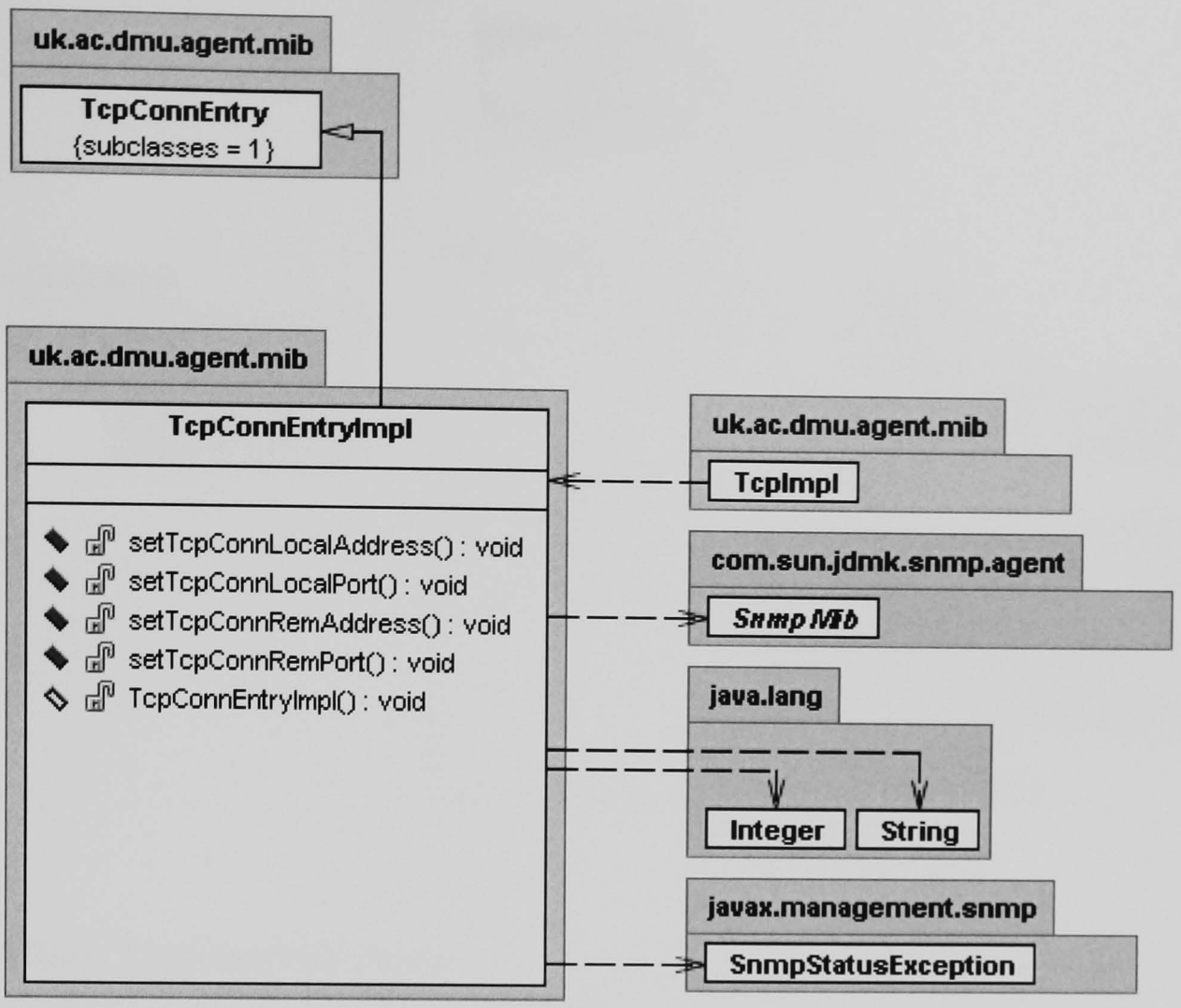
Class: Tcp



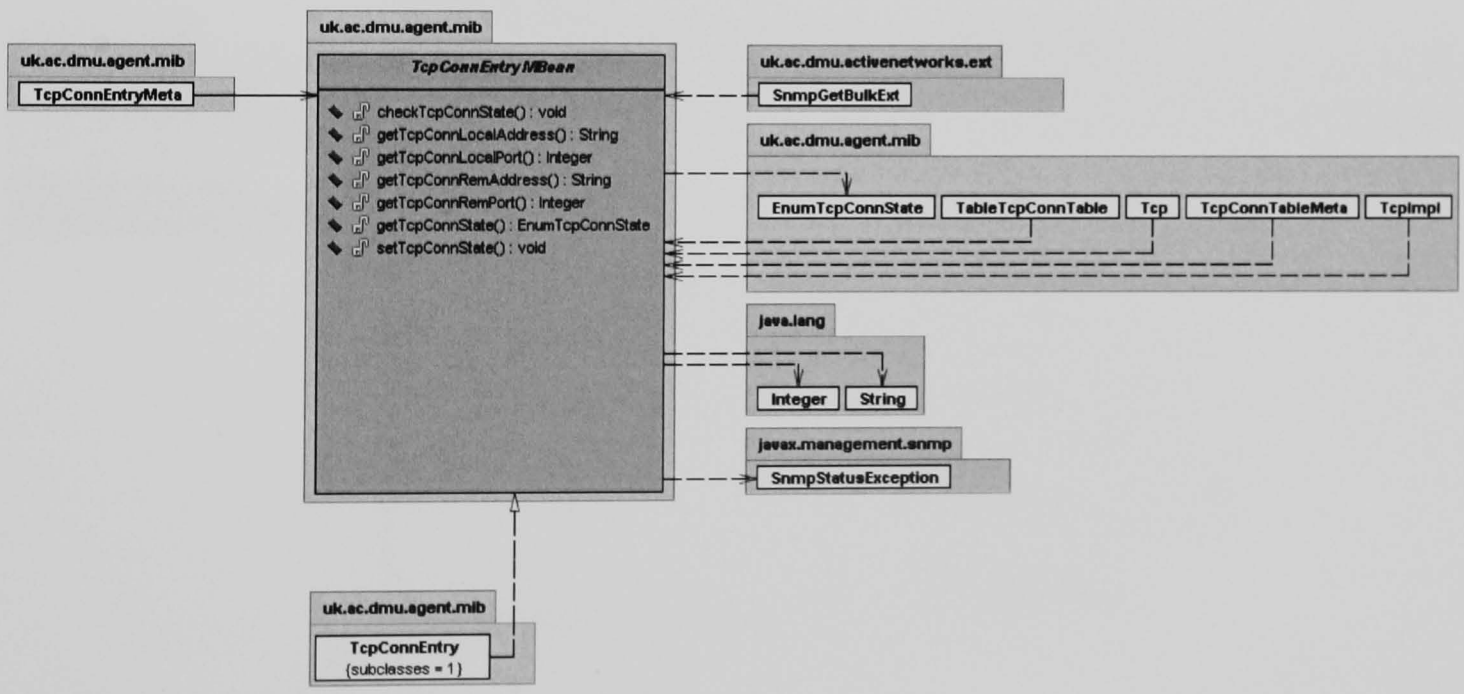
Class: TcpConnEntry



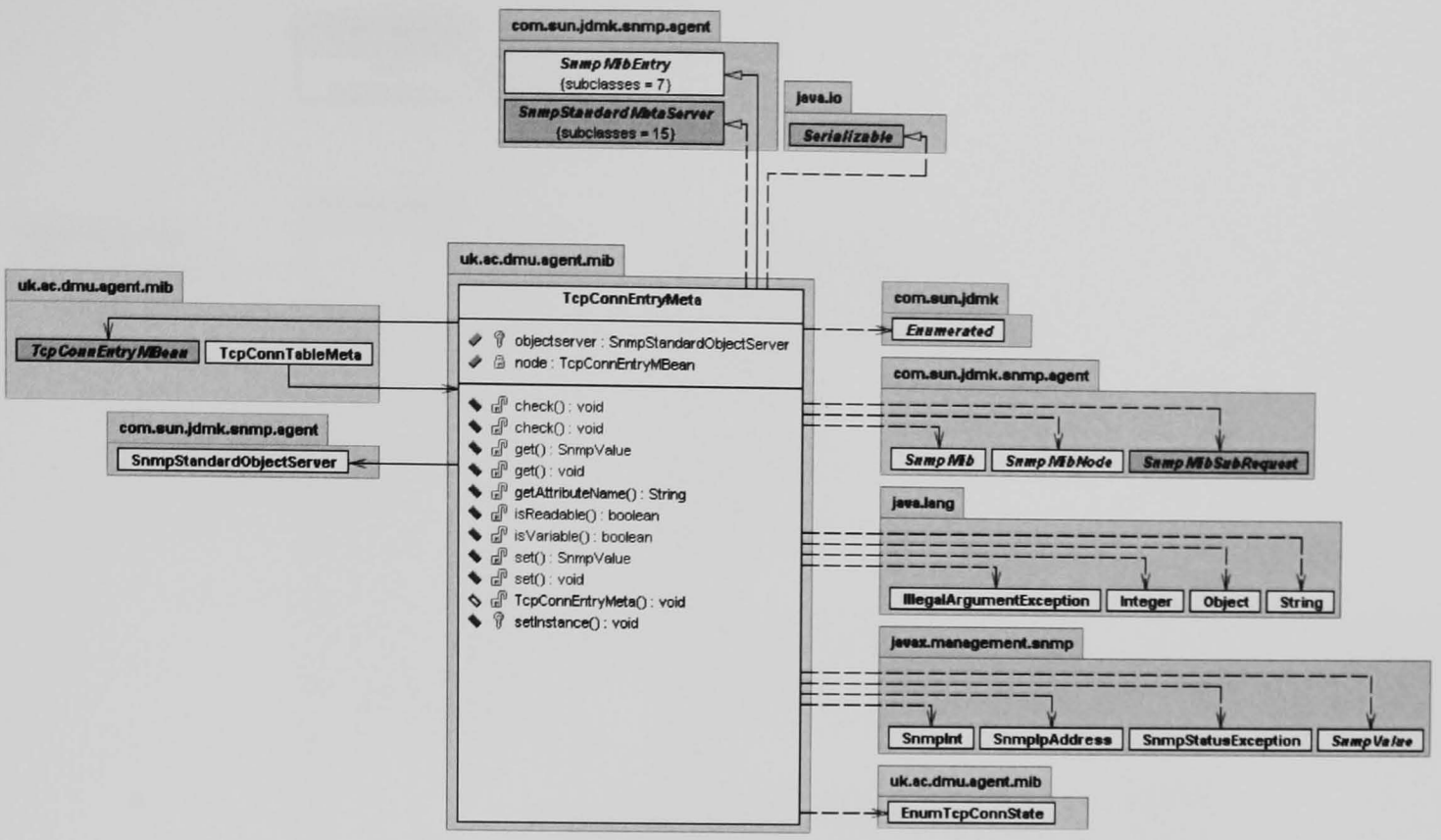
Class: TcpConnEntryImpl



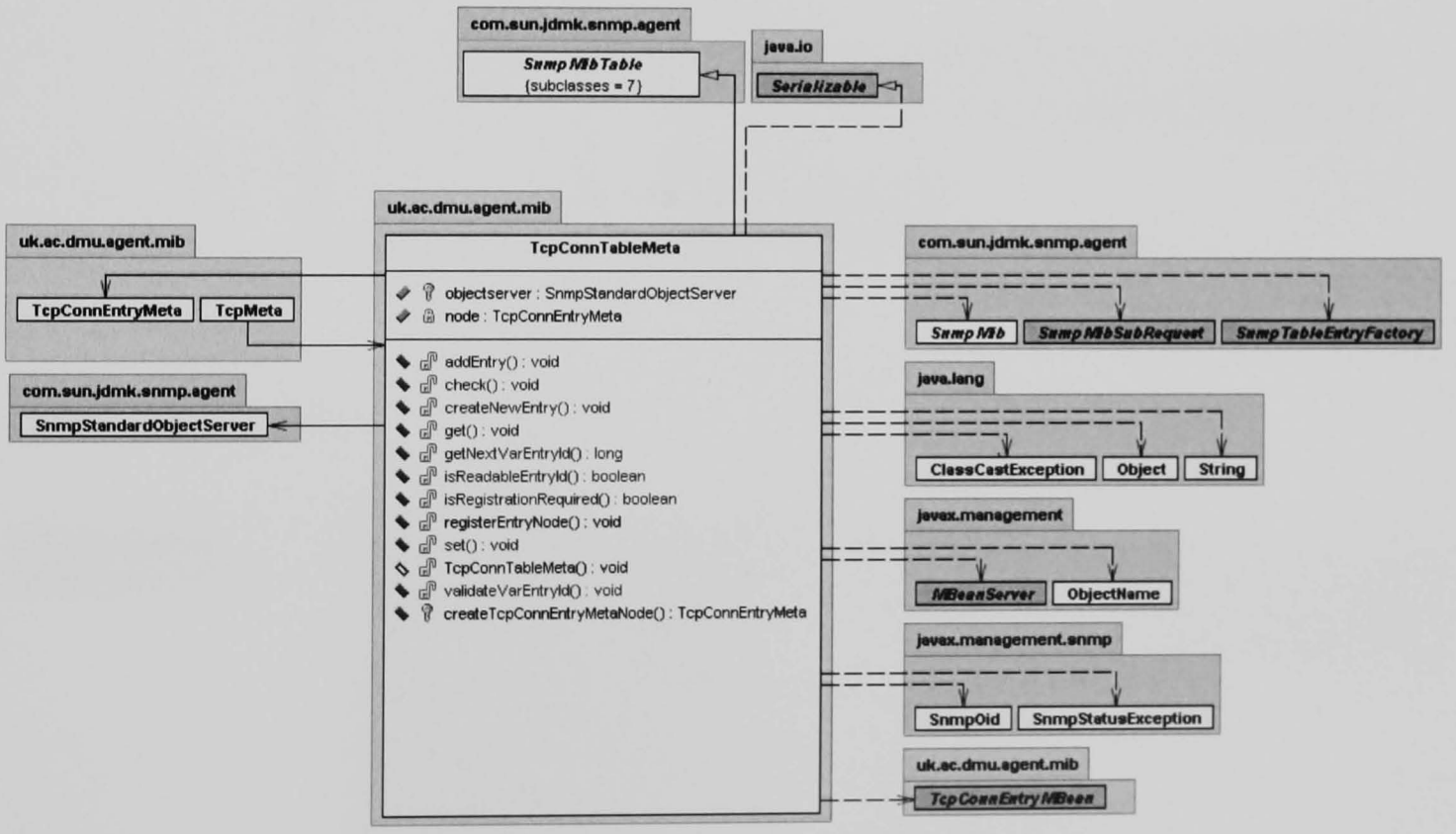
Interface: TcpConnEntryMBean



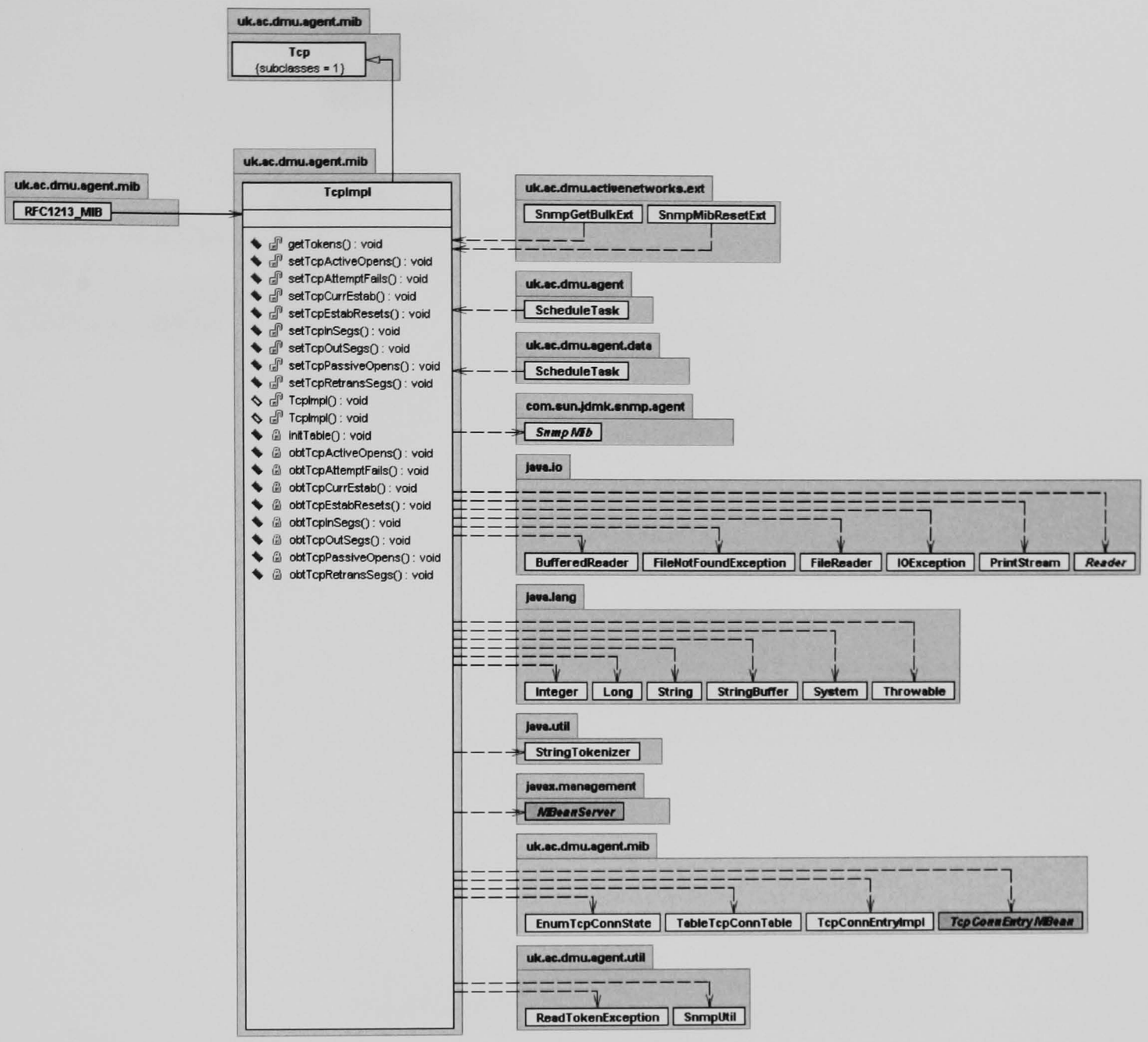
Class: TcpConnEntryMeta



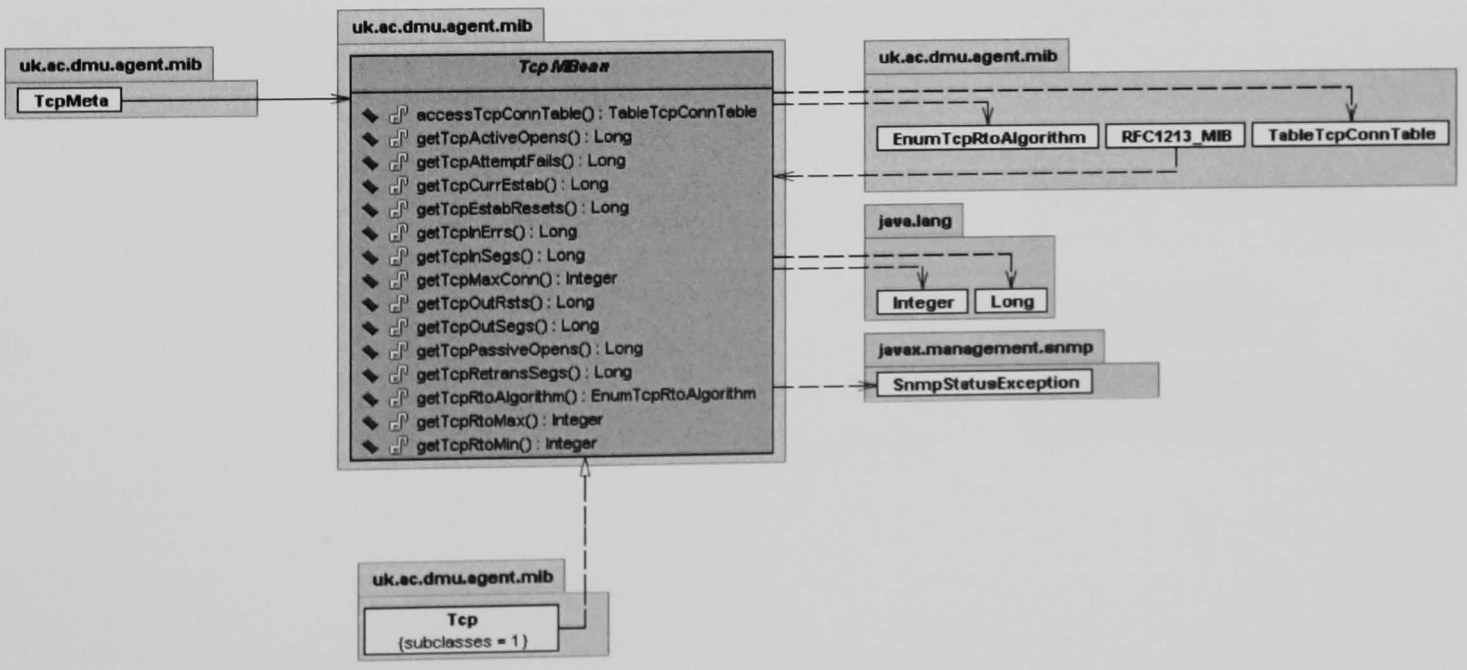
Class: TcpConnTableMeta



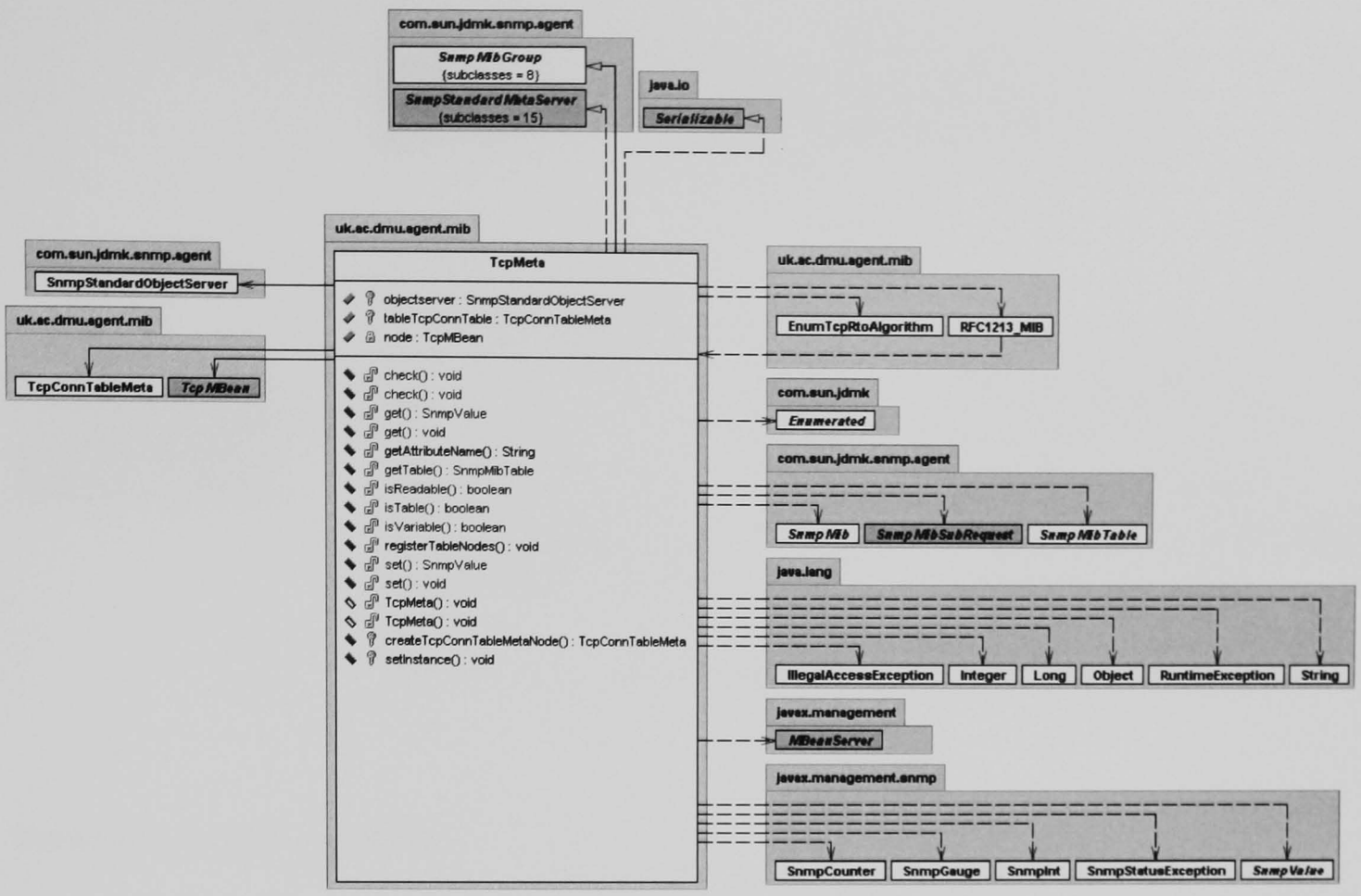
Class: TcpImpl



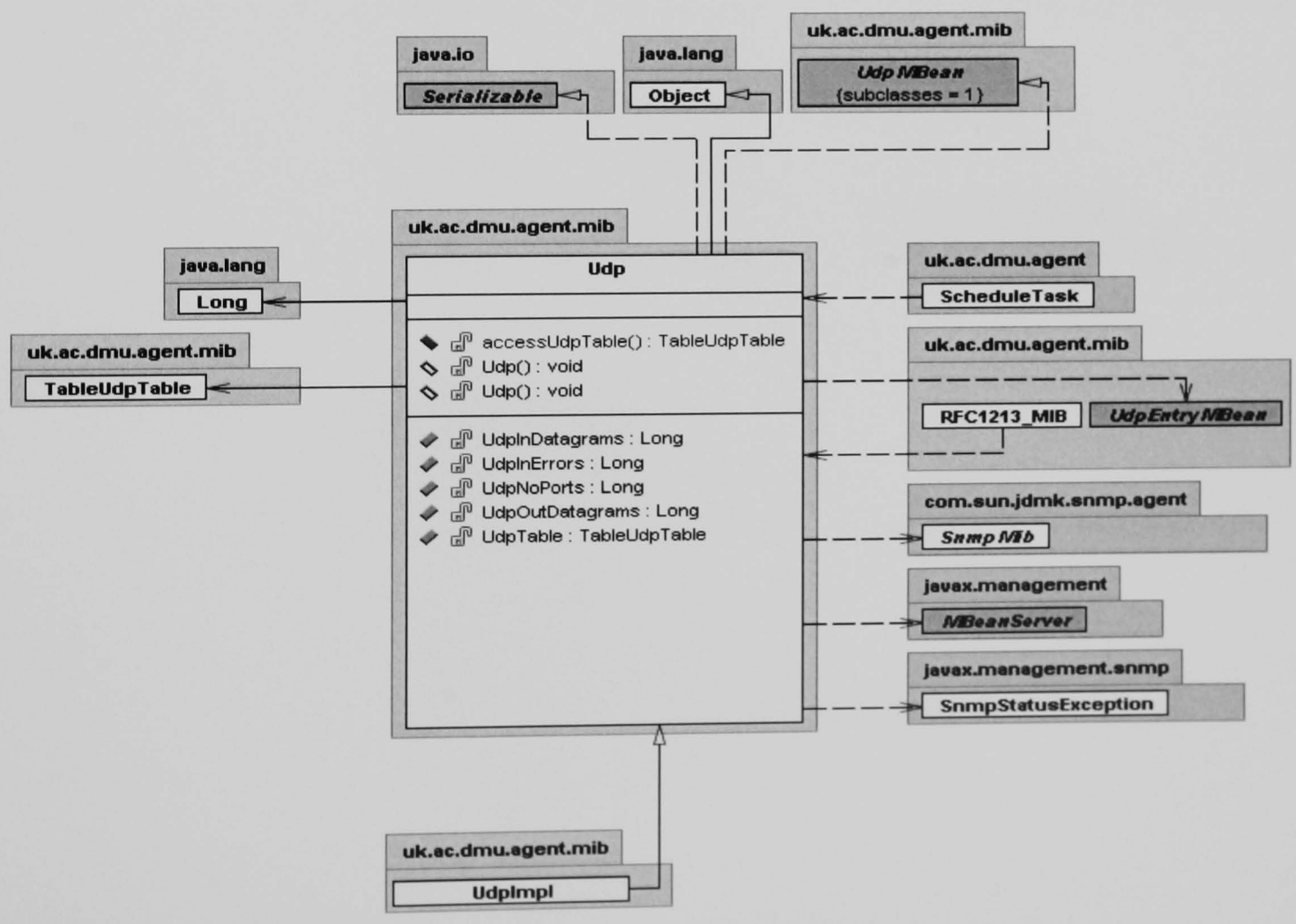
Interface: TcpMBean



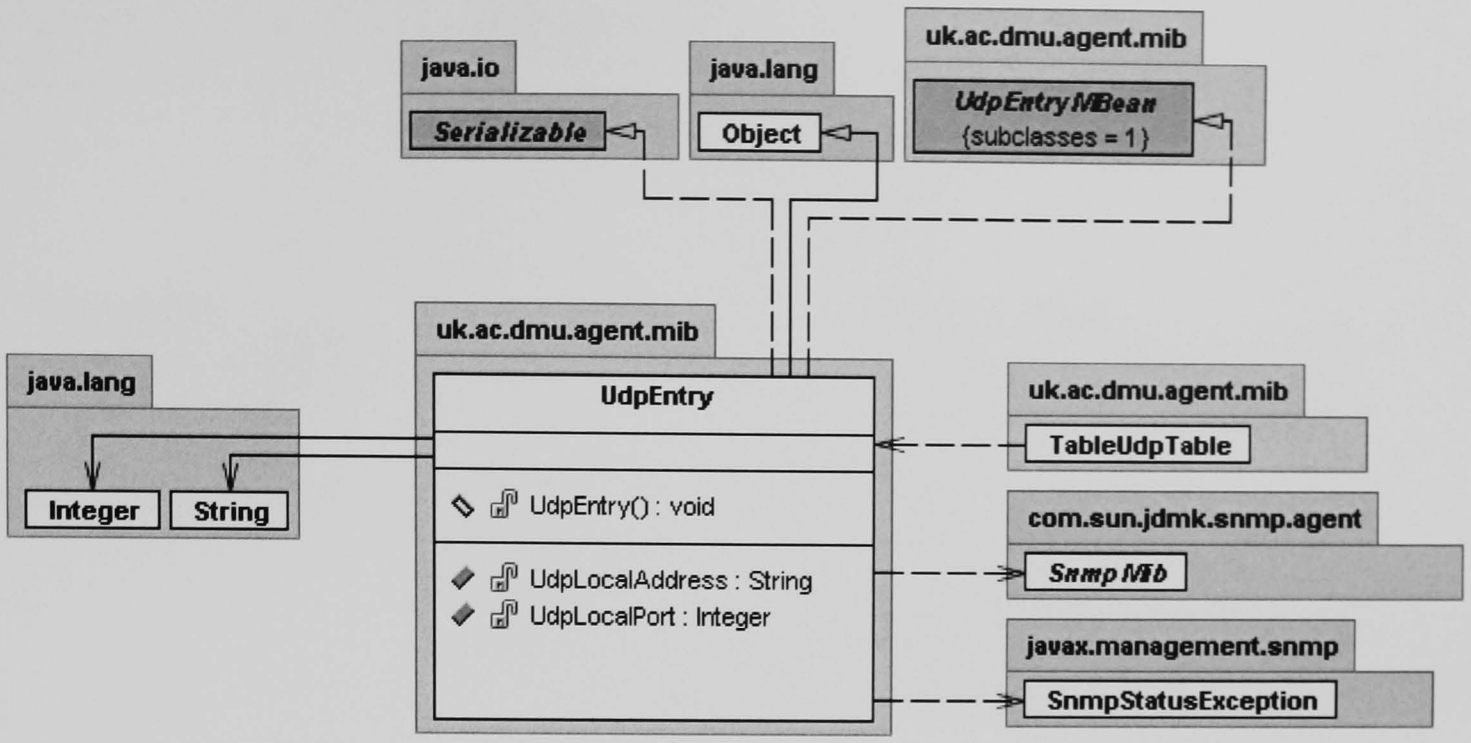
Class TcpMeta



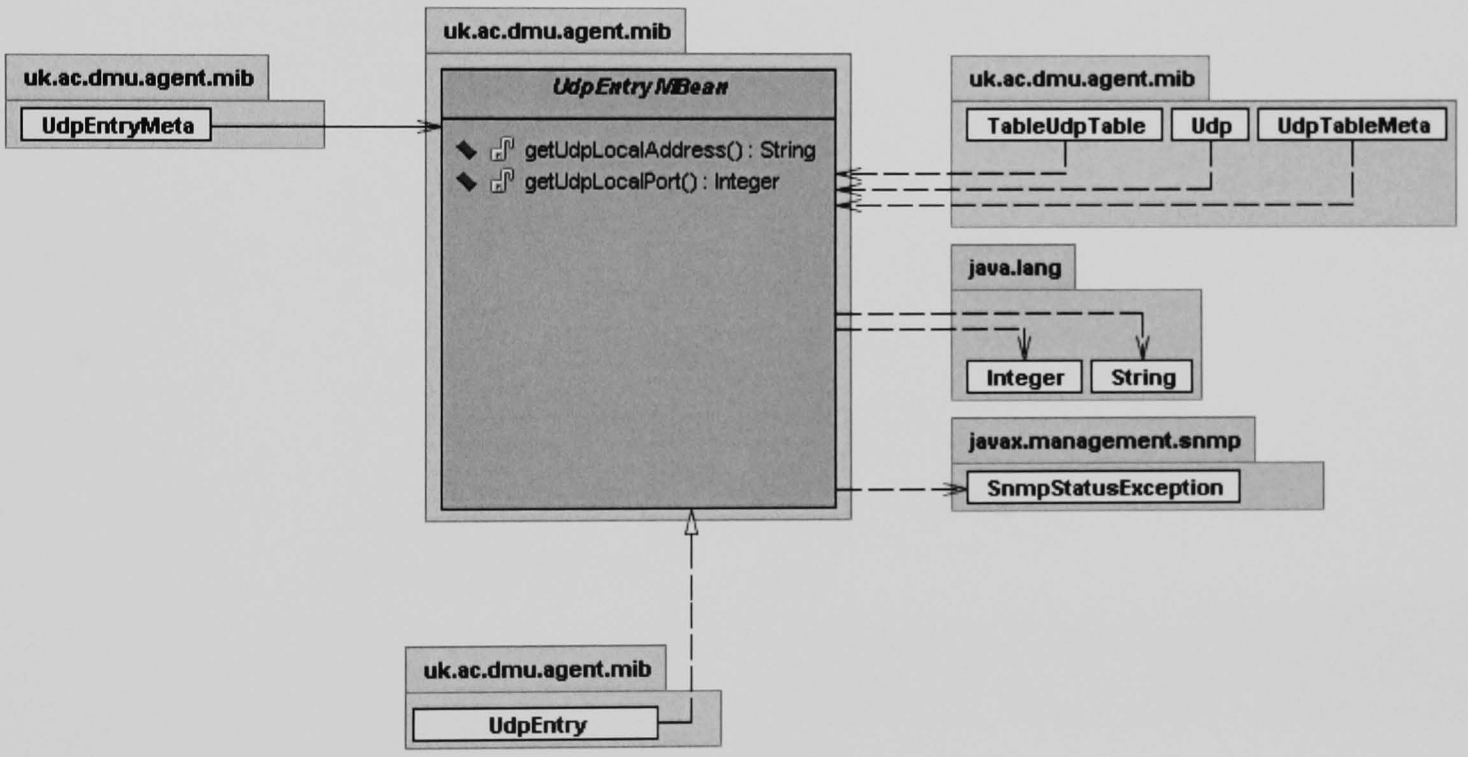
Class Udp



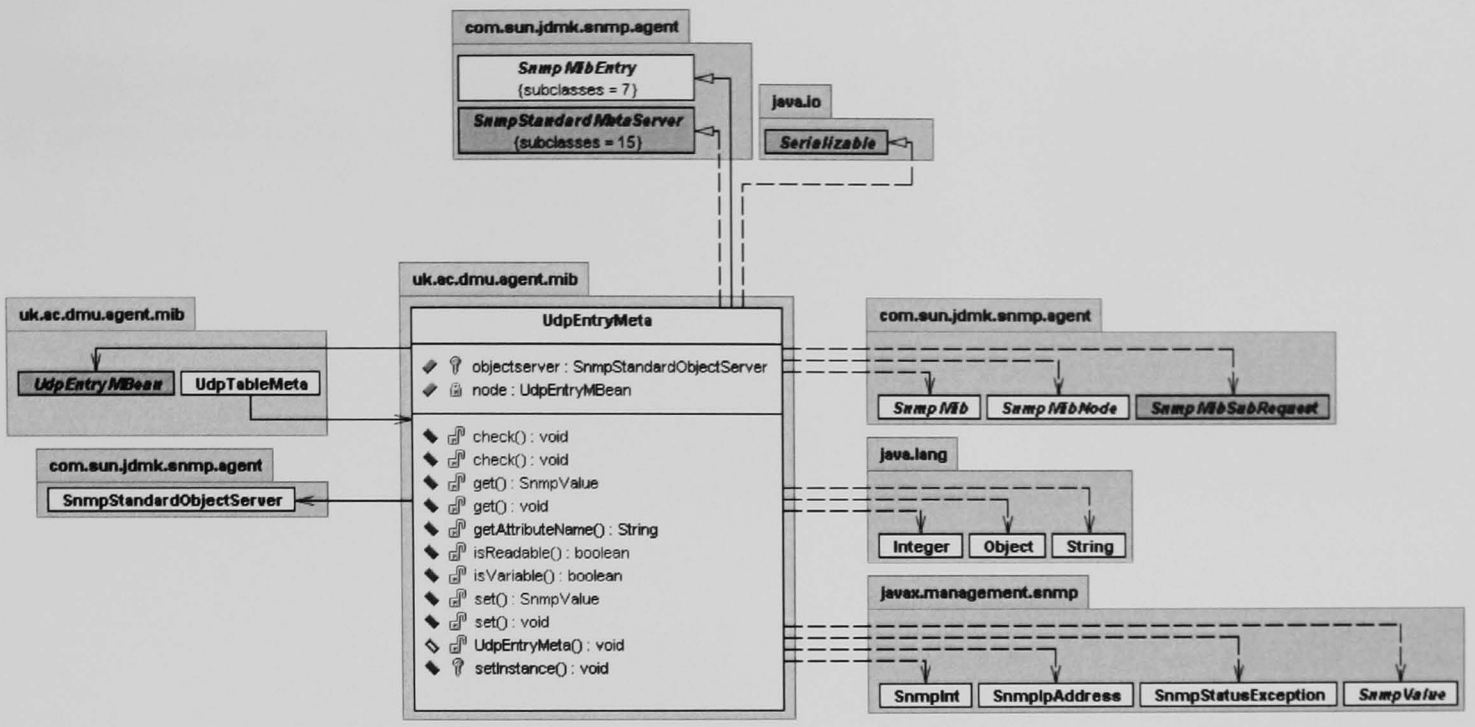
Class: UdpEntry



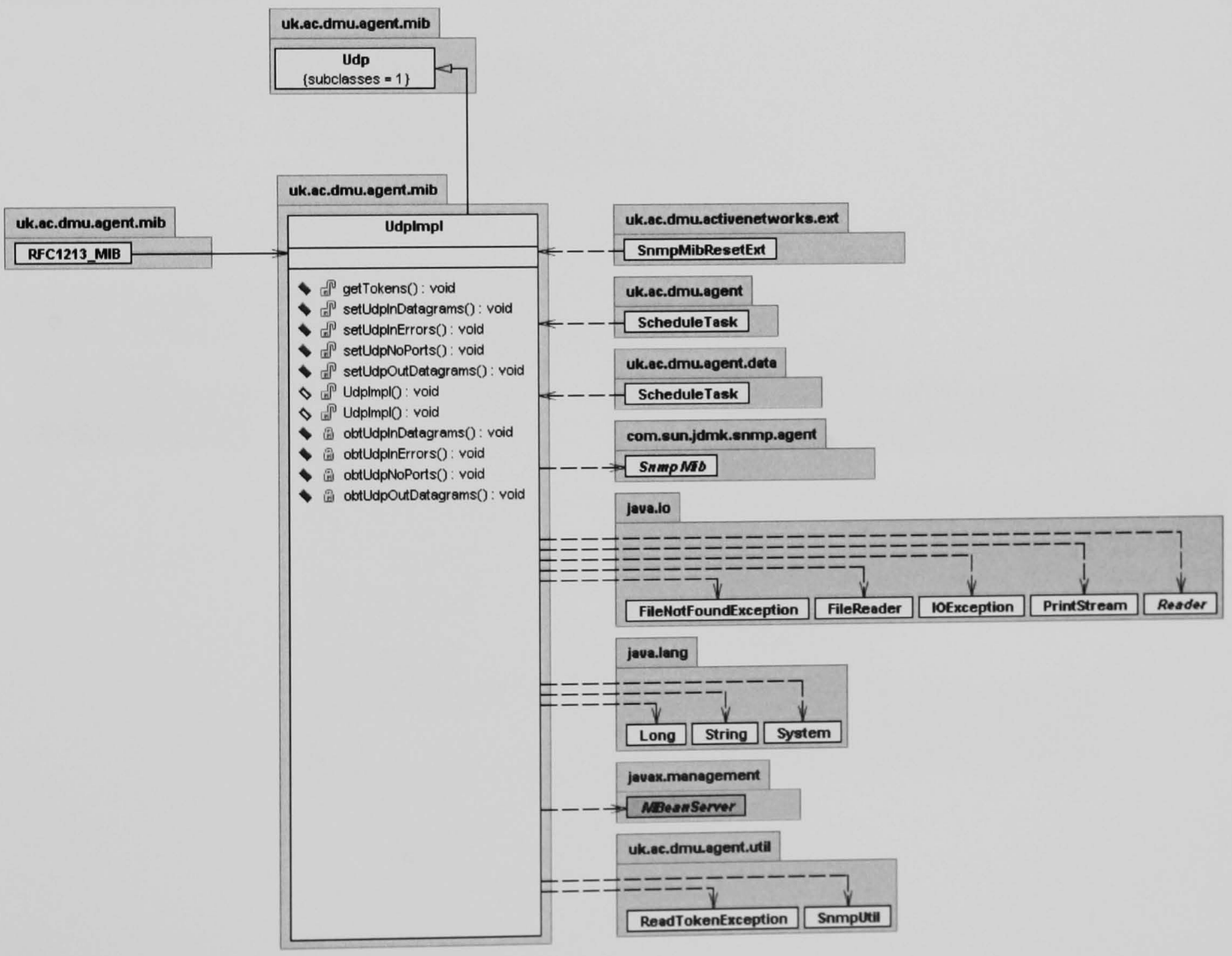
Interface: UdpEntryMBean



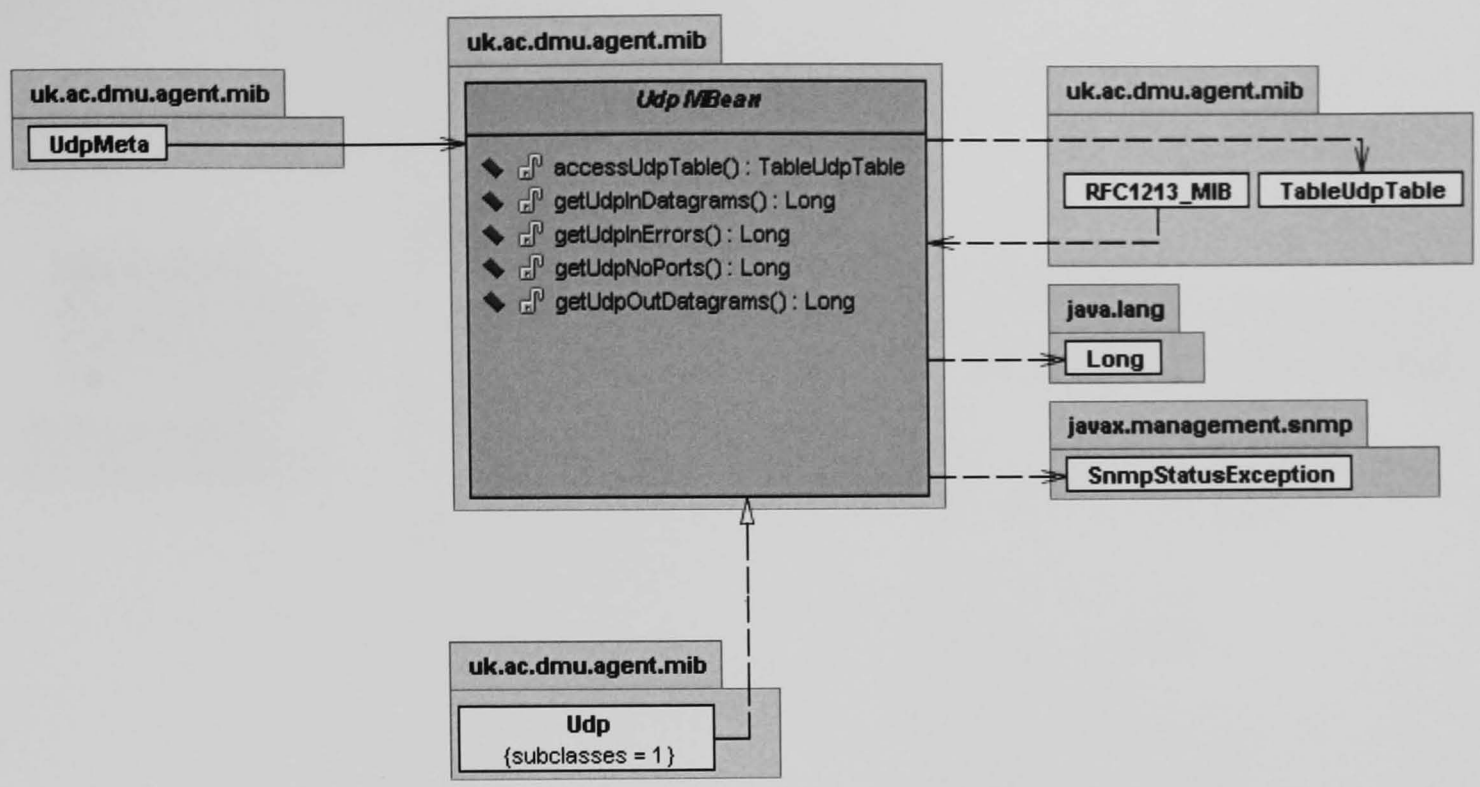
Class: UdpEntryMeta



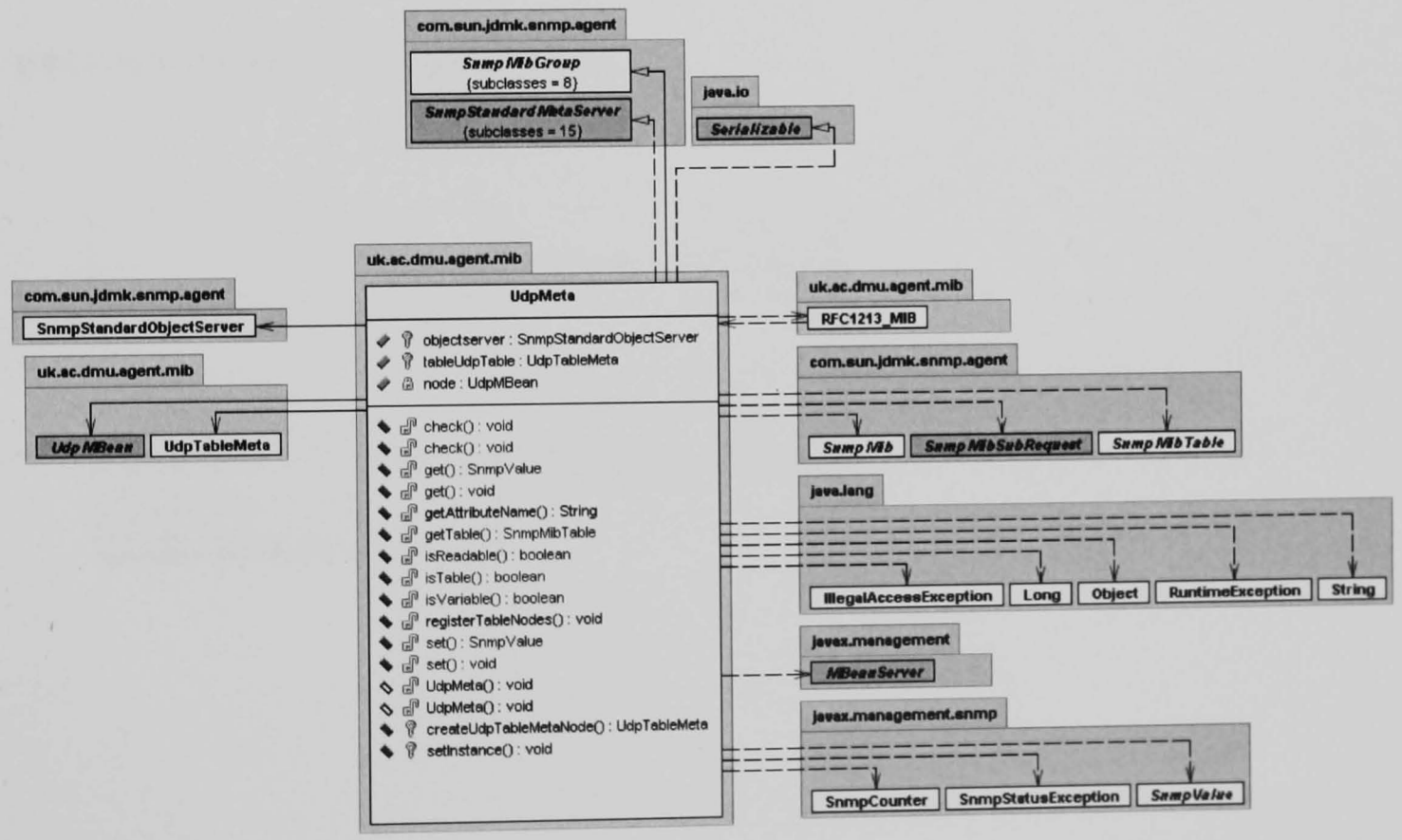
Class: UdpImpl



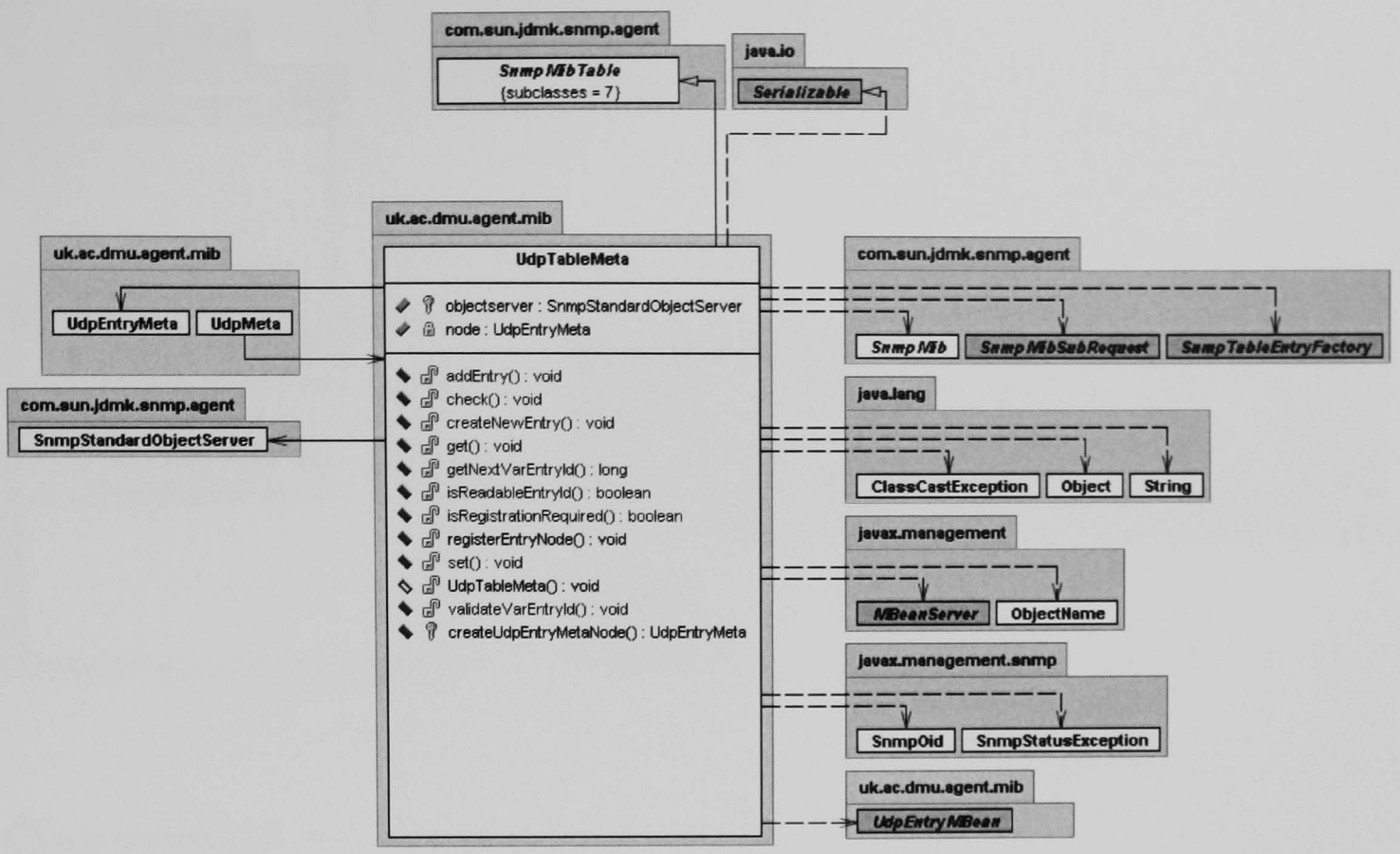
Interface: UdpMBean



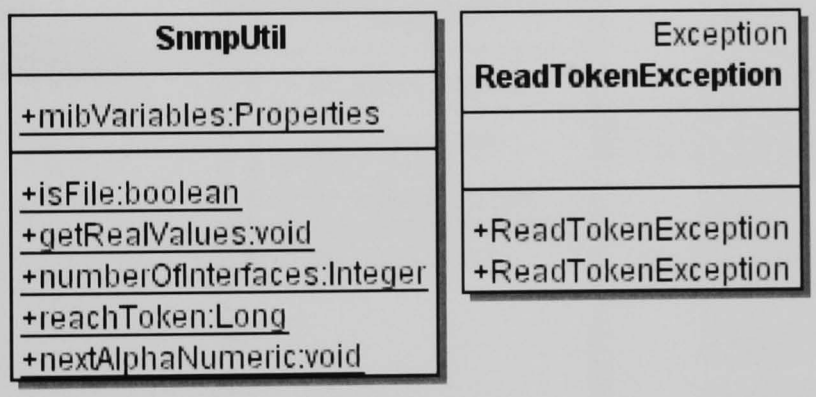
Class: UdpMeta



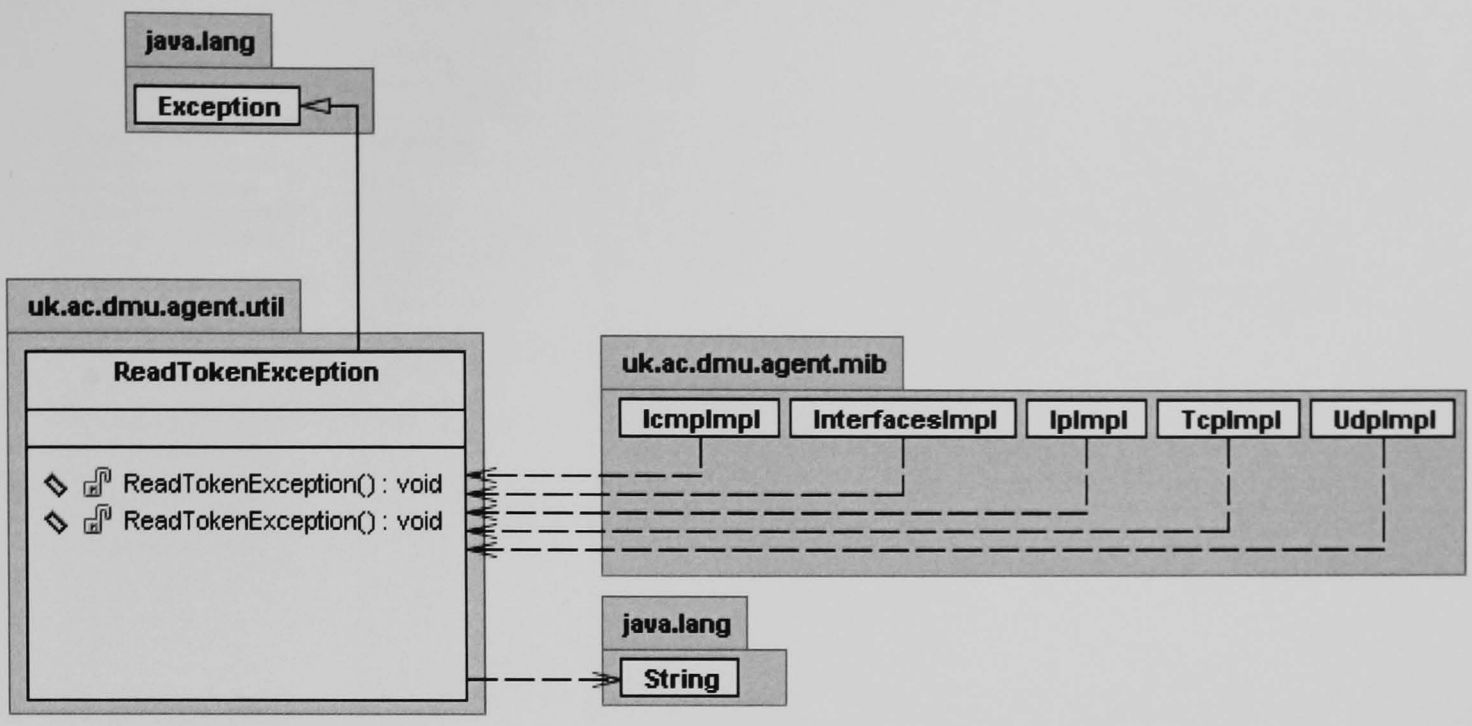
Class: UdpTableMeta



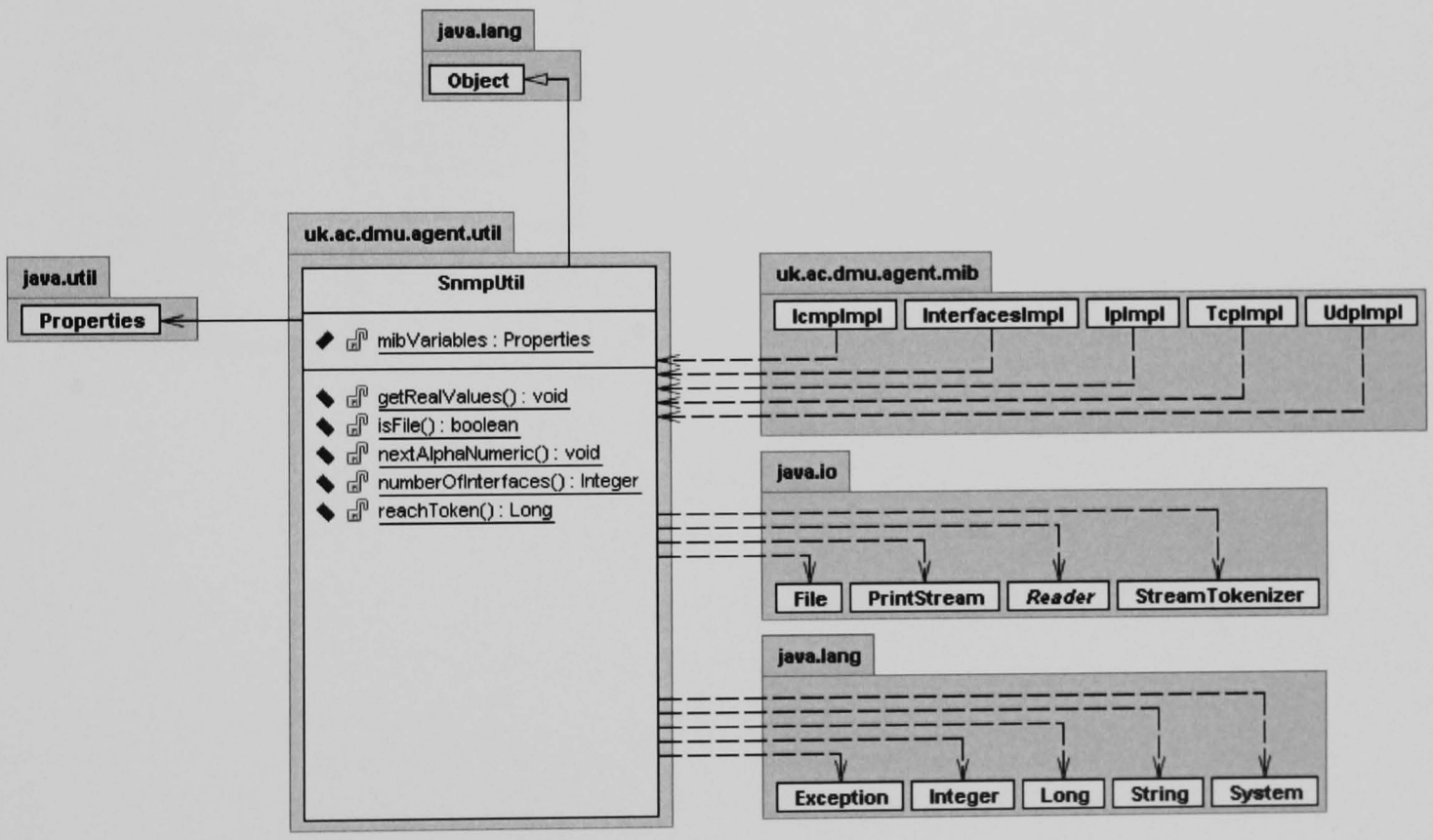
package uk.ac.dmu.agent.util



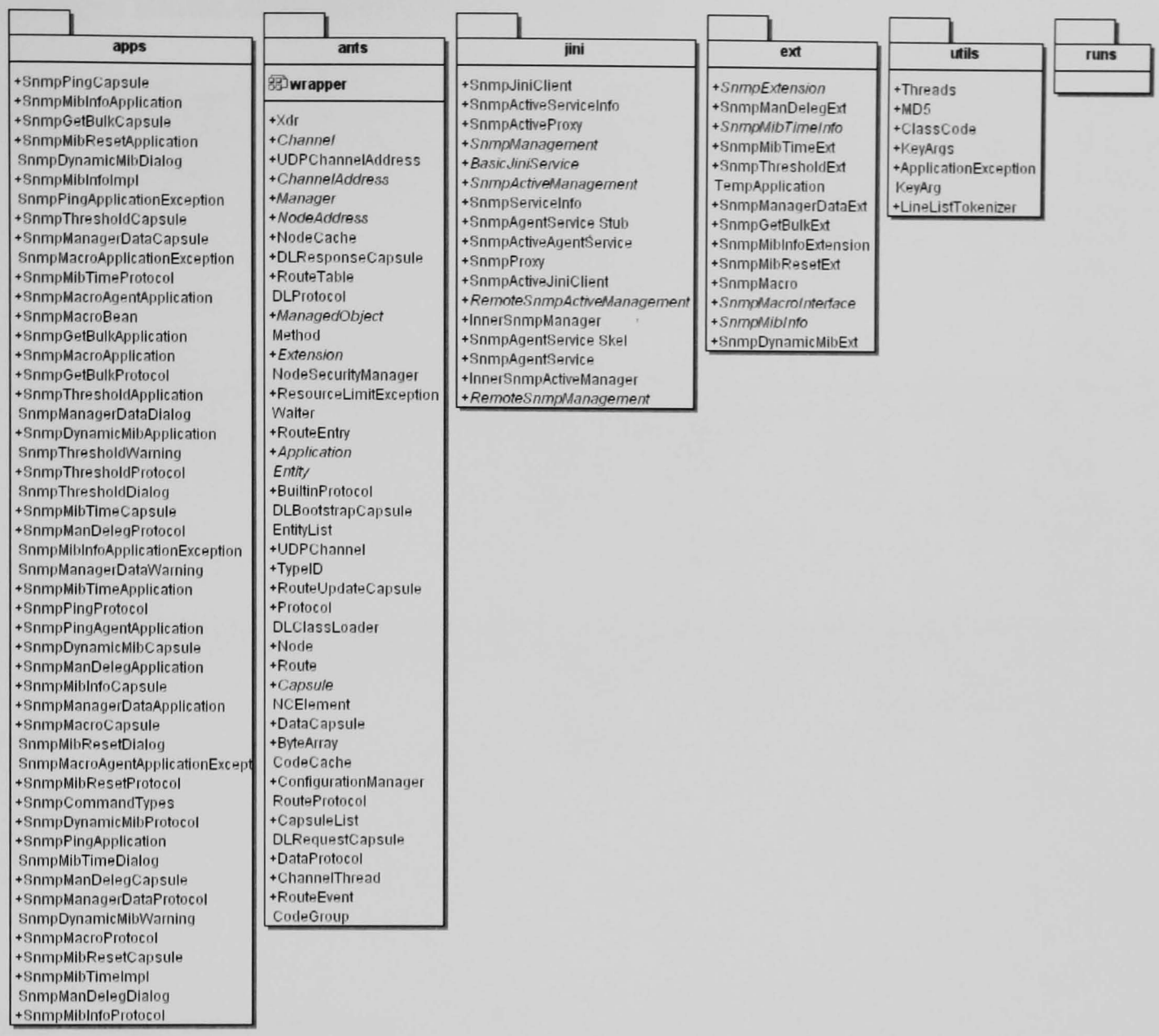
Class: ReadTokenException



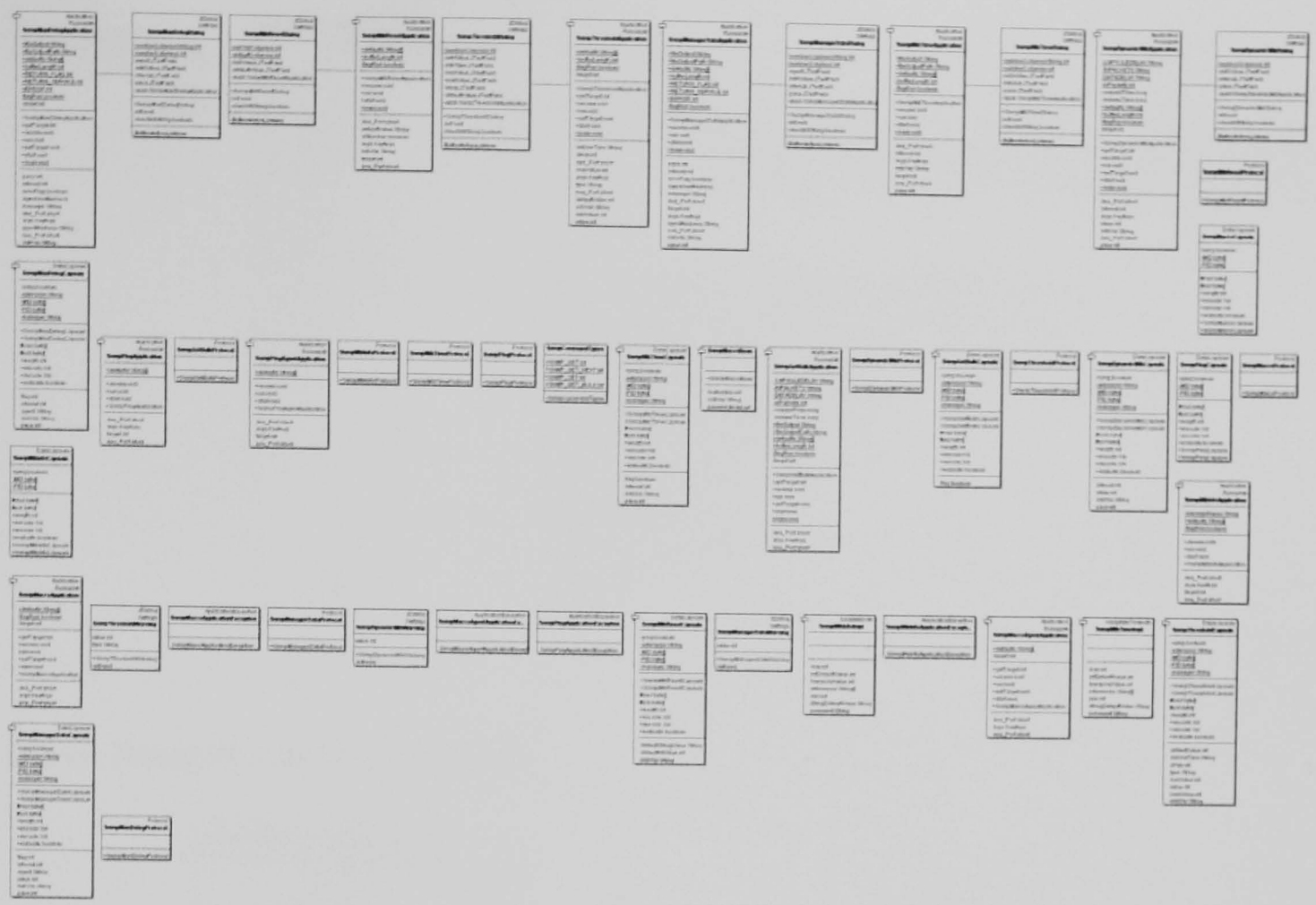
Class: SnmpUtil



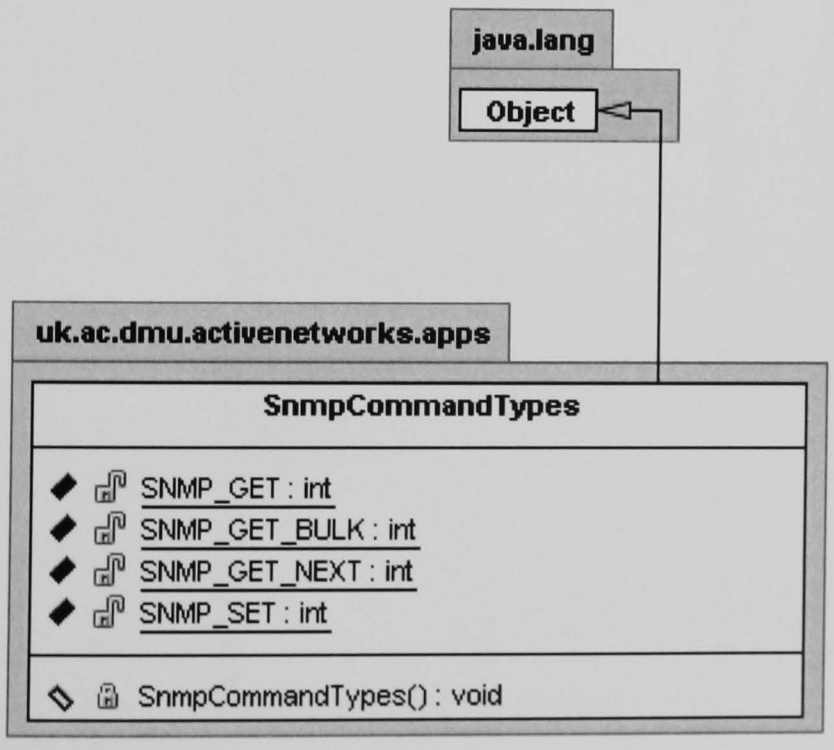
C.3 PACKAGE:UK.AC.DMU.ACTIVENETWORKS



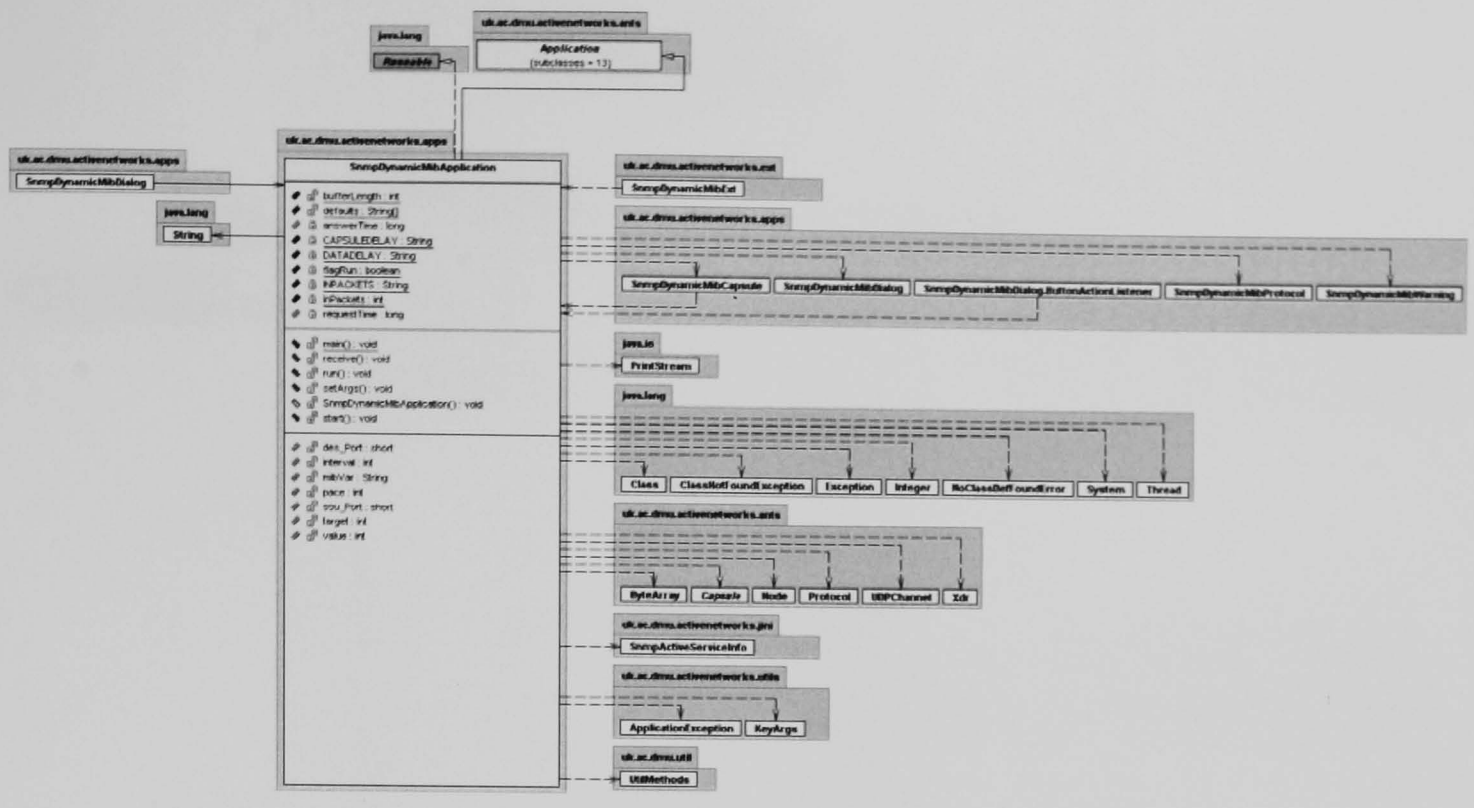
Package: uk.ac.dmu.activenetworks.apps



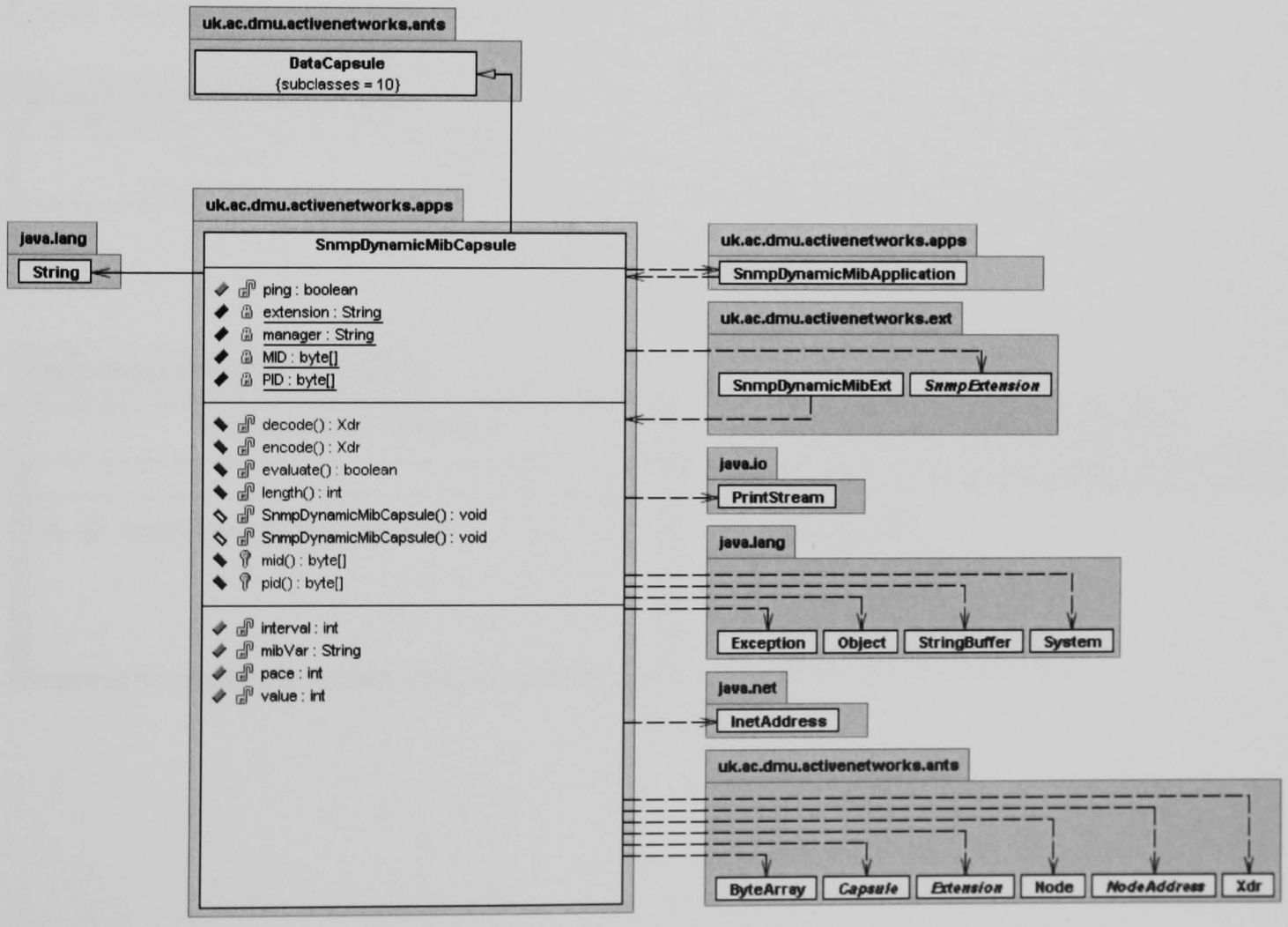
Class: SnmpCommandTypes



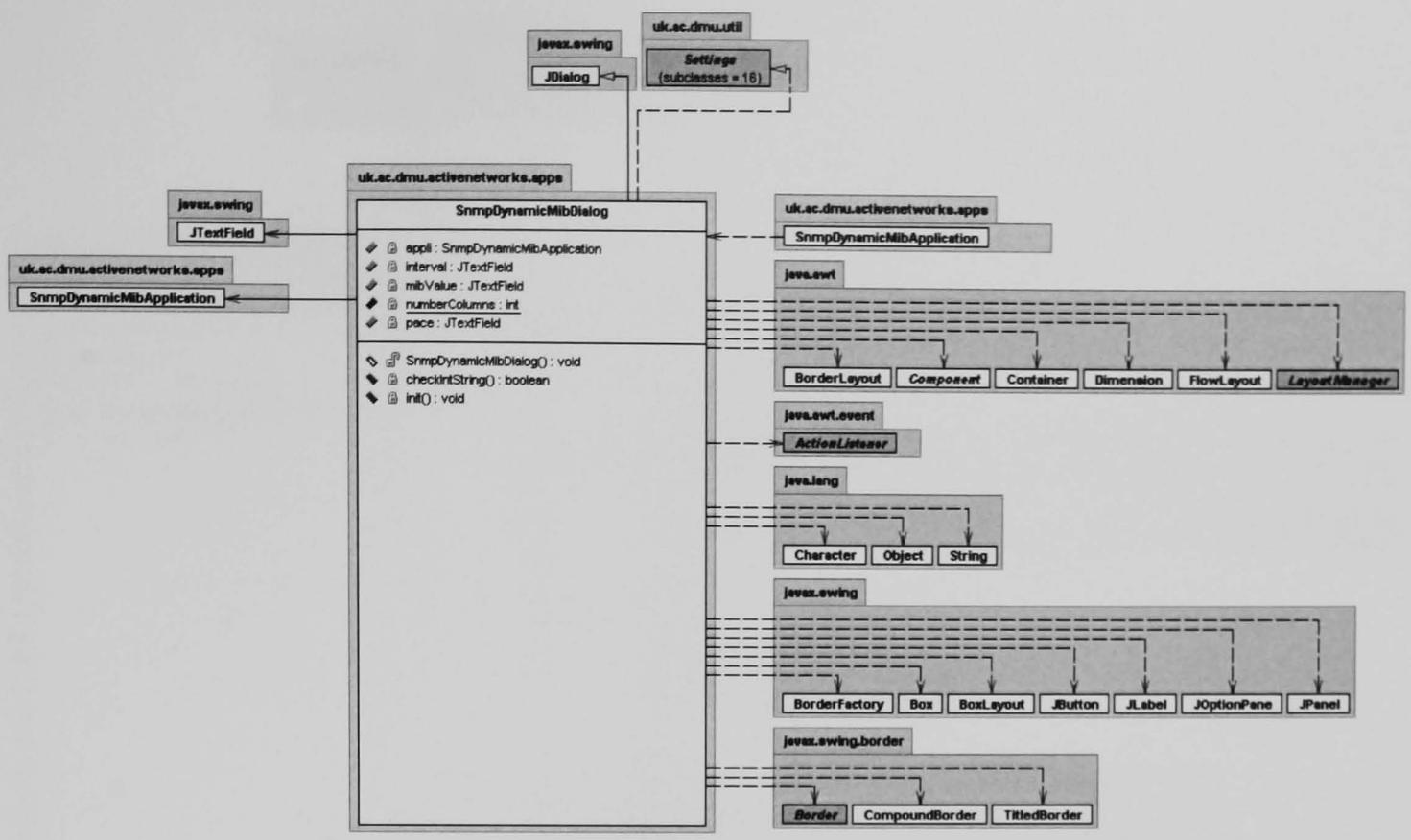
Class:SnmpDynamicMibApplication



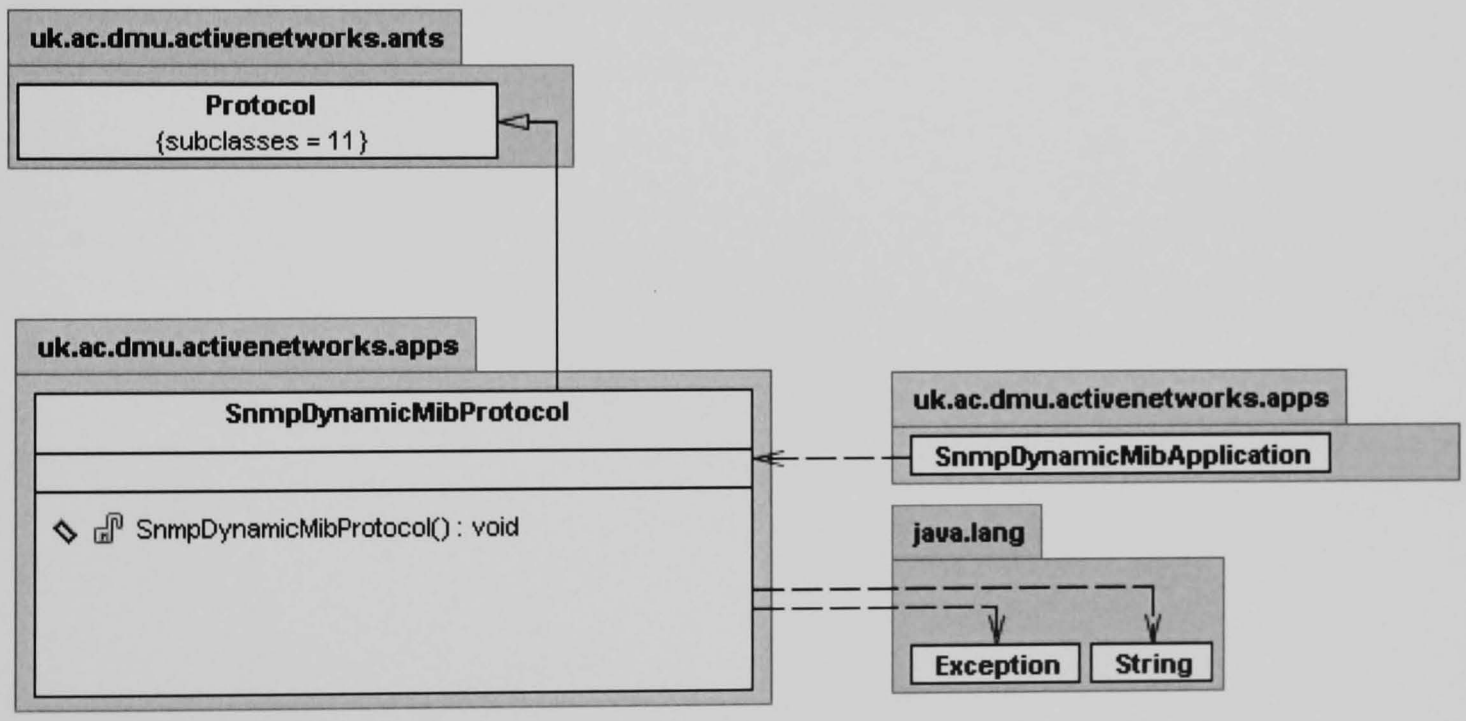
Class: SnmpDynamicMibCapsule



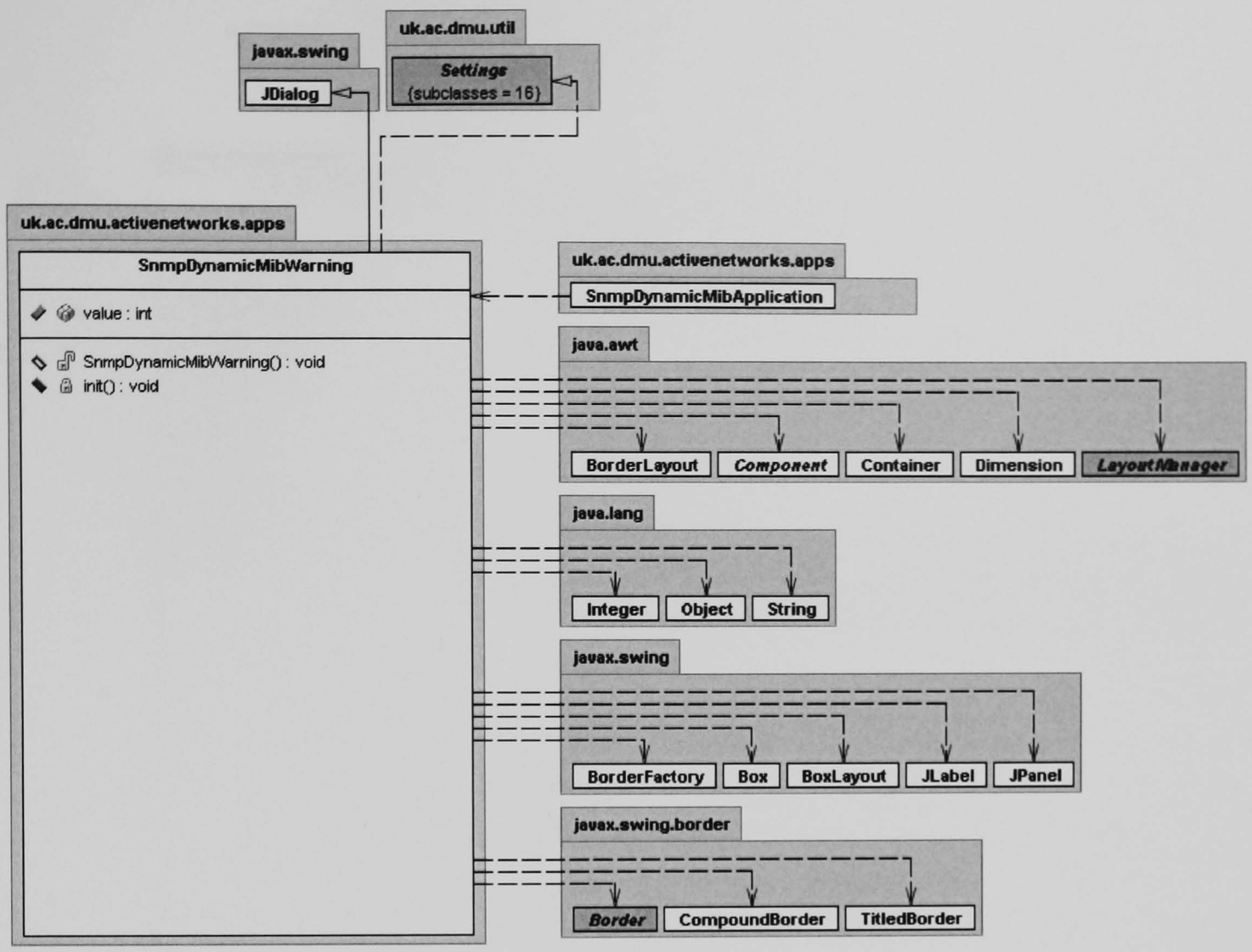
Class: SnmpDynamicMibDialog



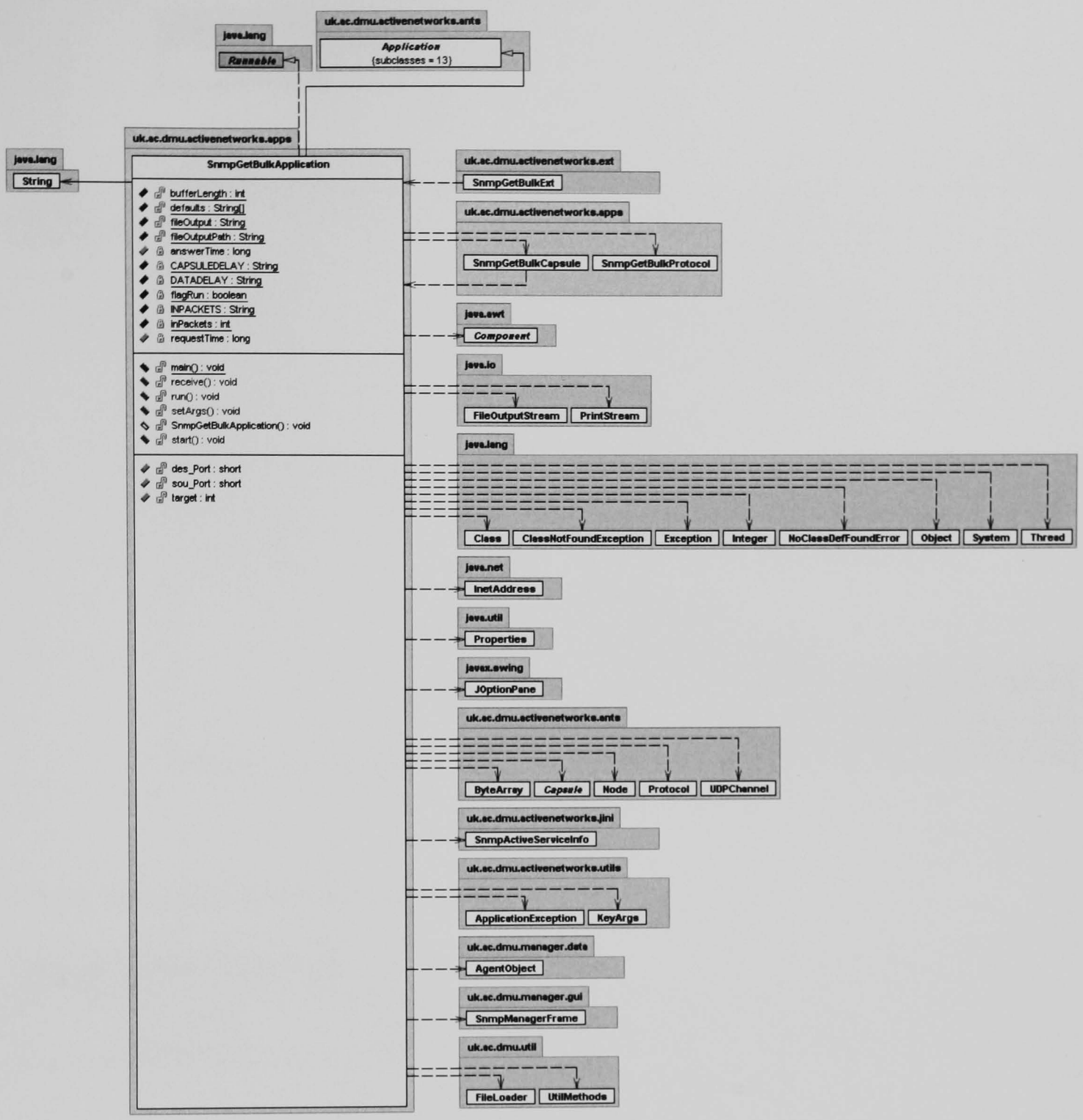
Class: SnmpDynamicMibProtocol



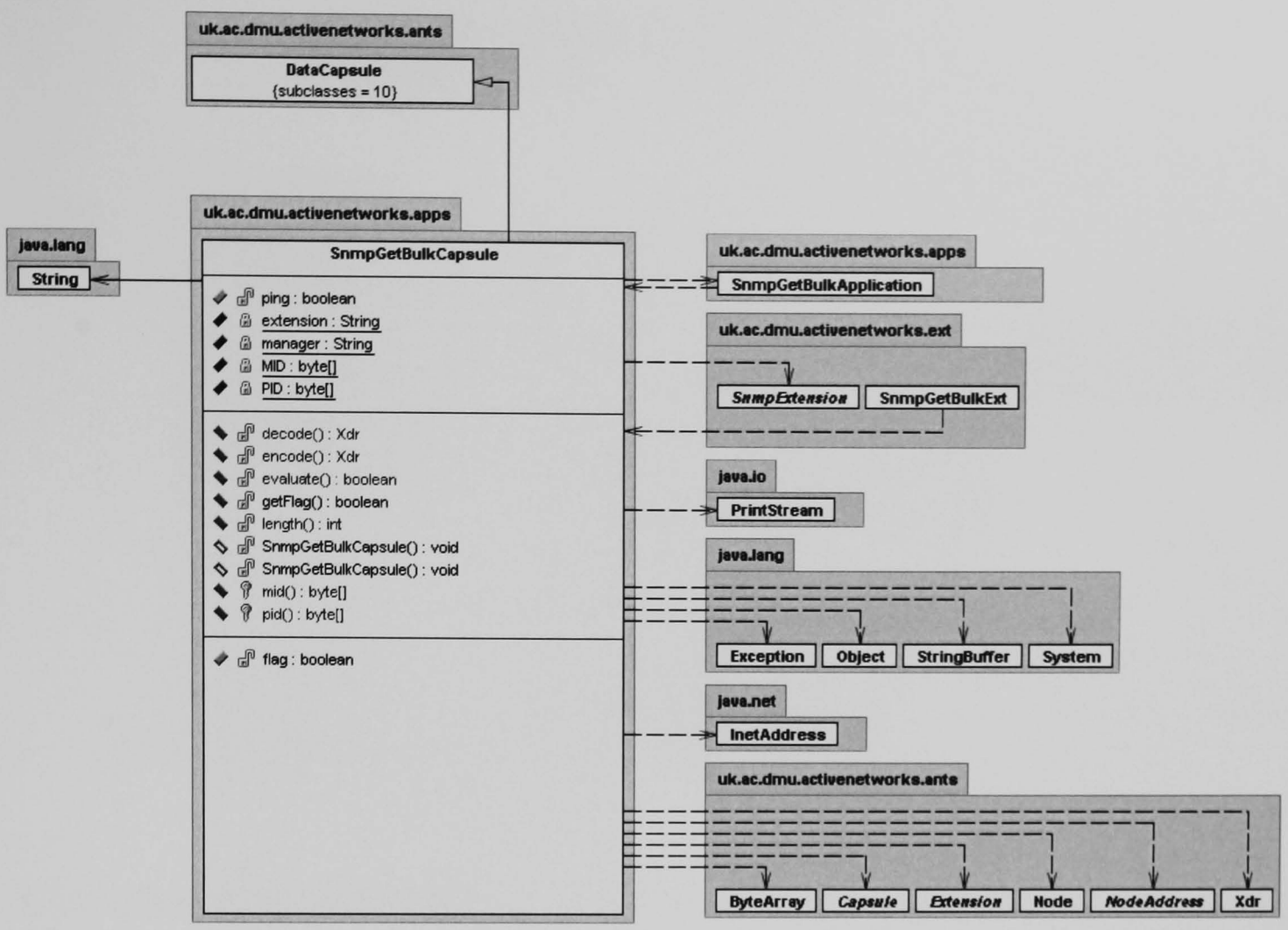
Class:SnmpDynamicMibWarning



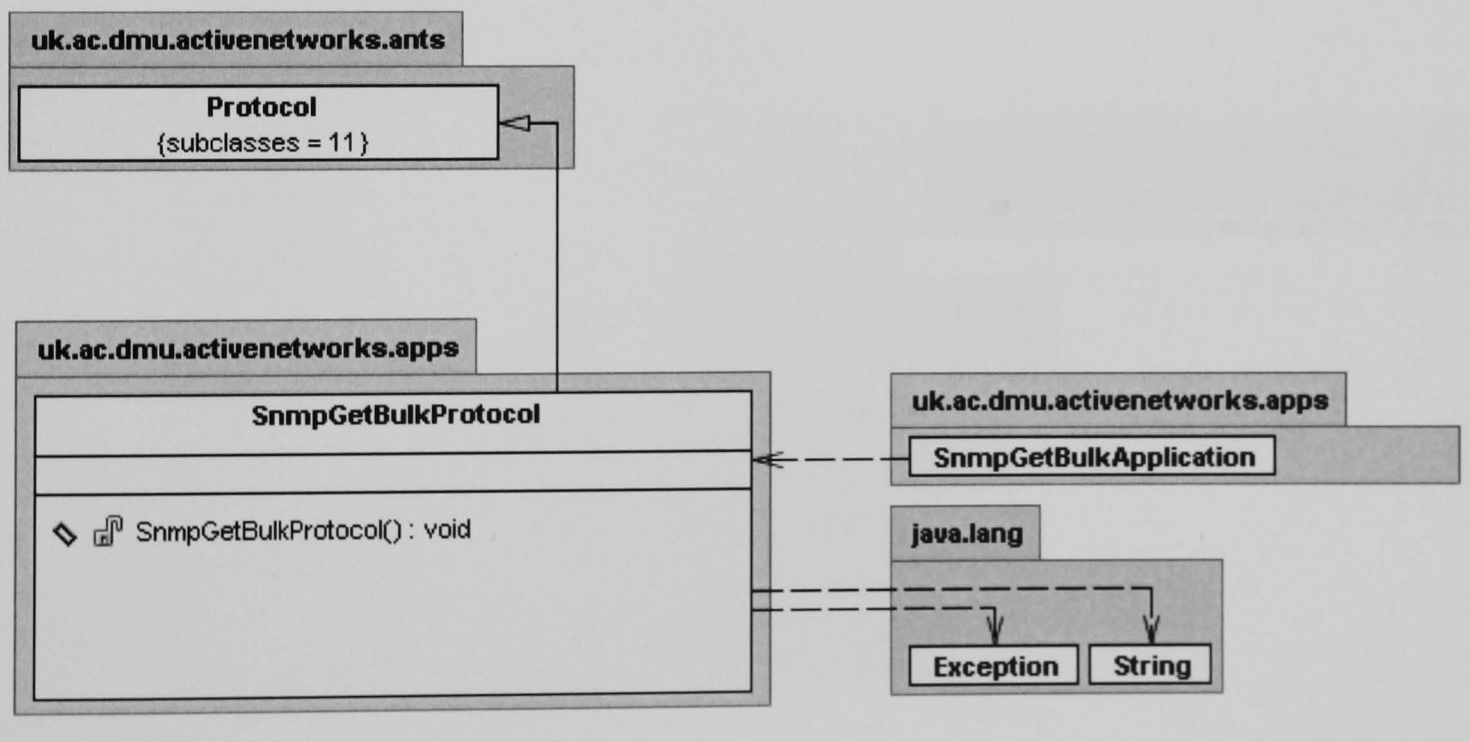
Class: SnmpGetBulkApplication



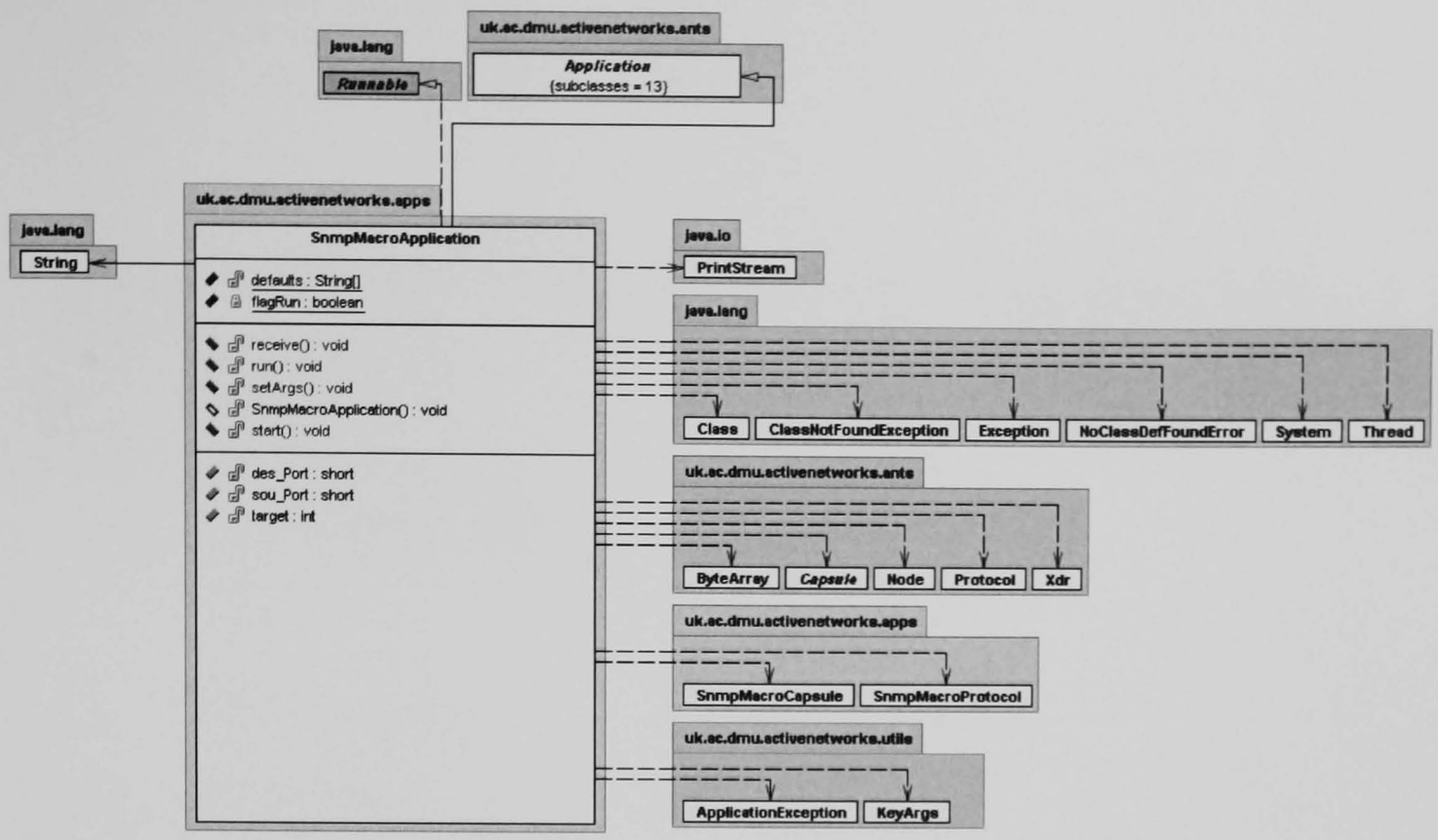
Class: SnmpGetBulkCapsule



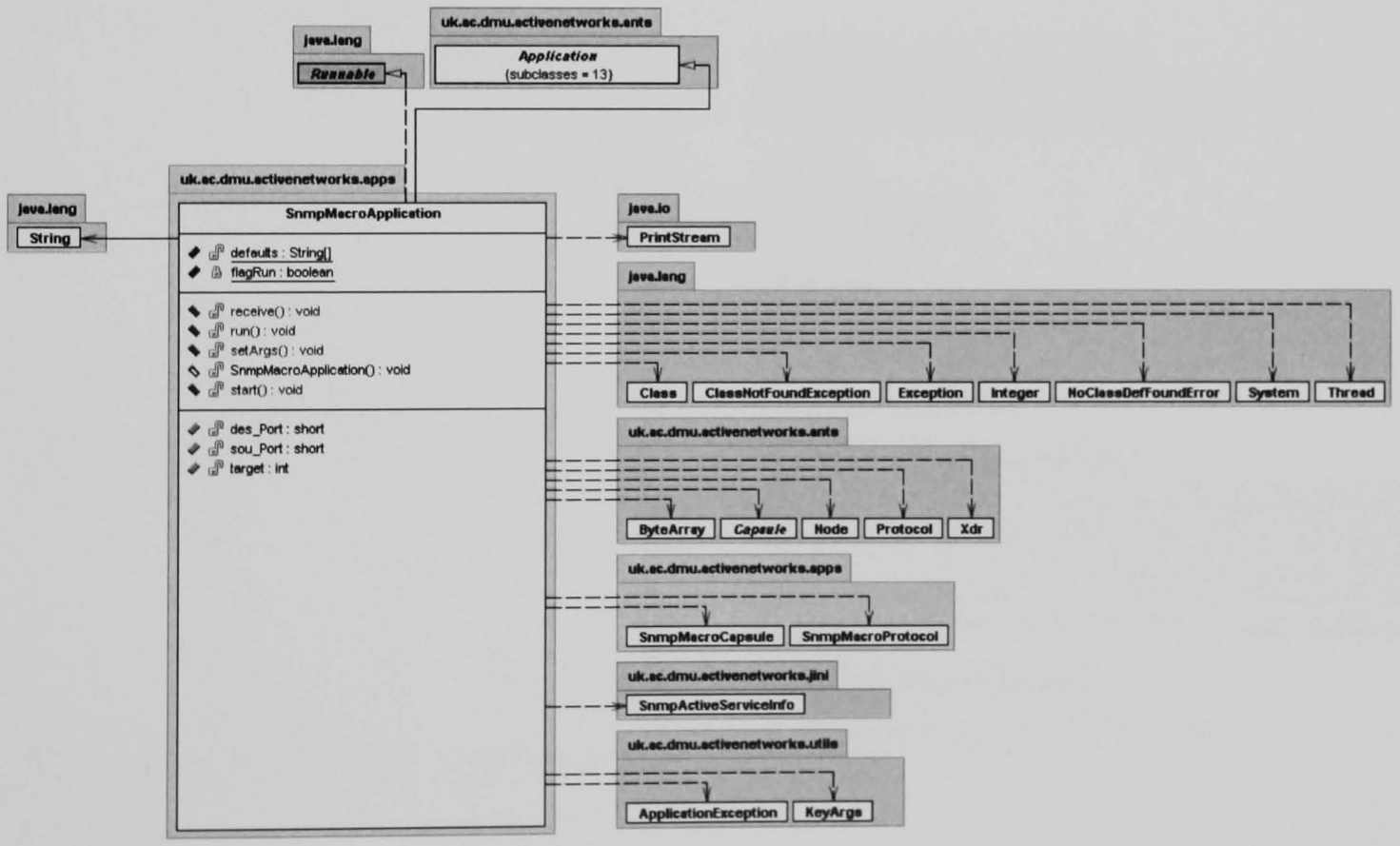
Class: SnmpGetBulkProtocol



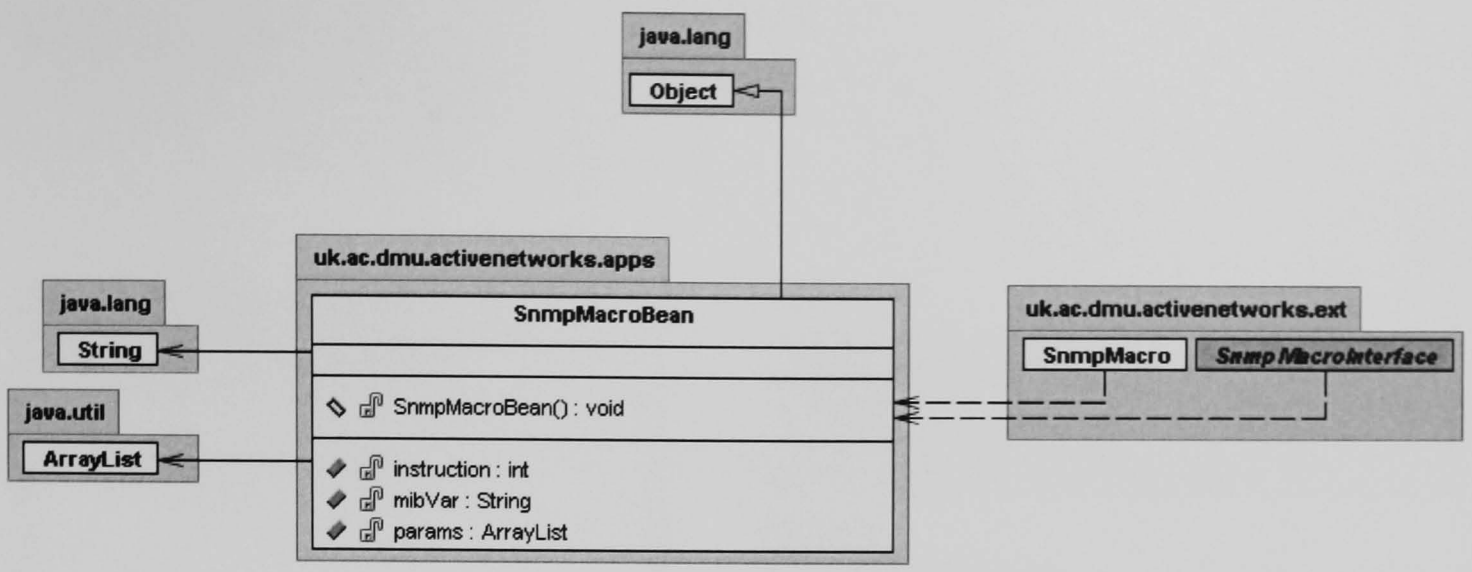
Class: SnmpMacroAgentApplication



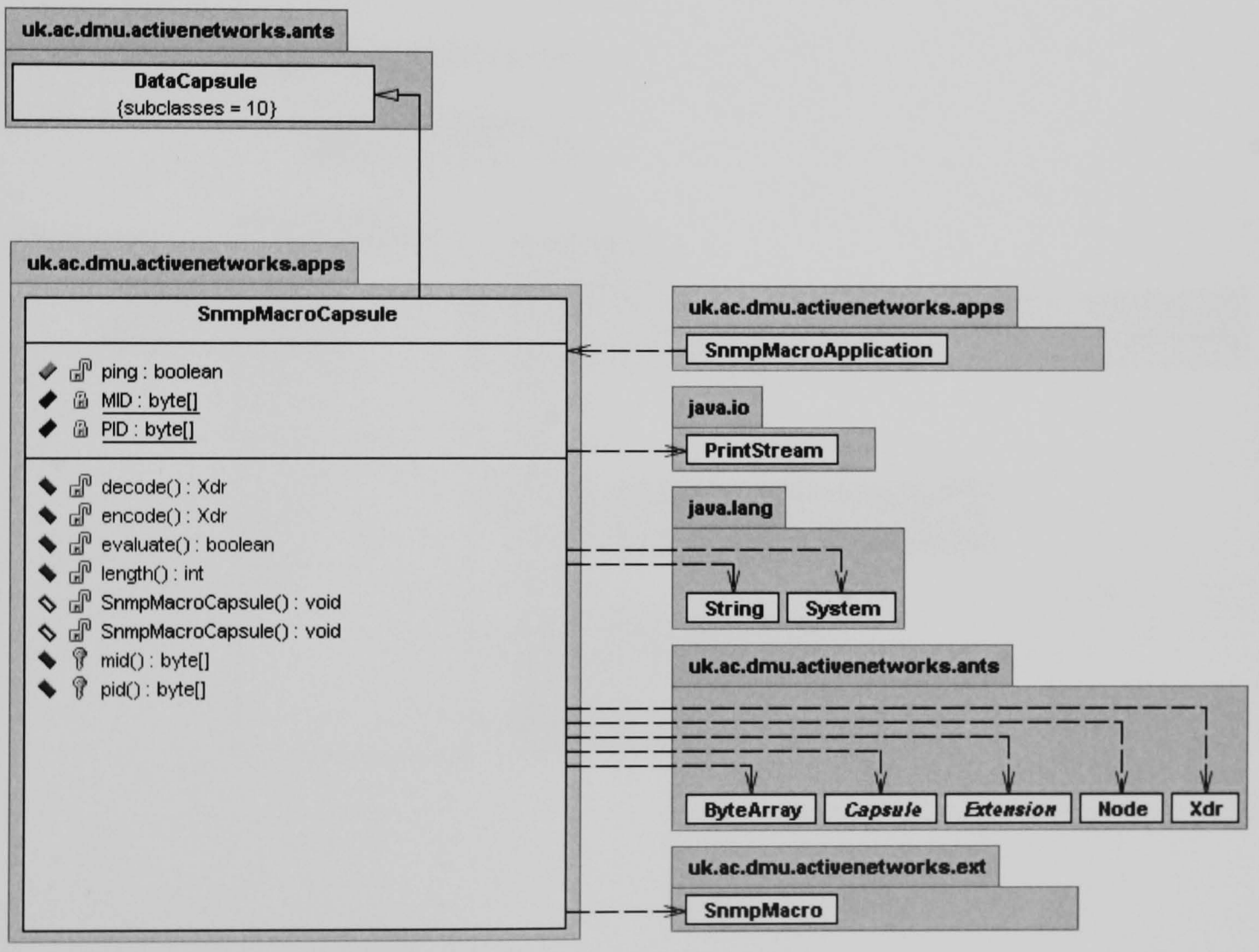
Class: SnmpMacroApplication



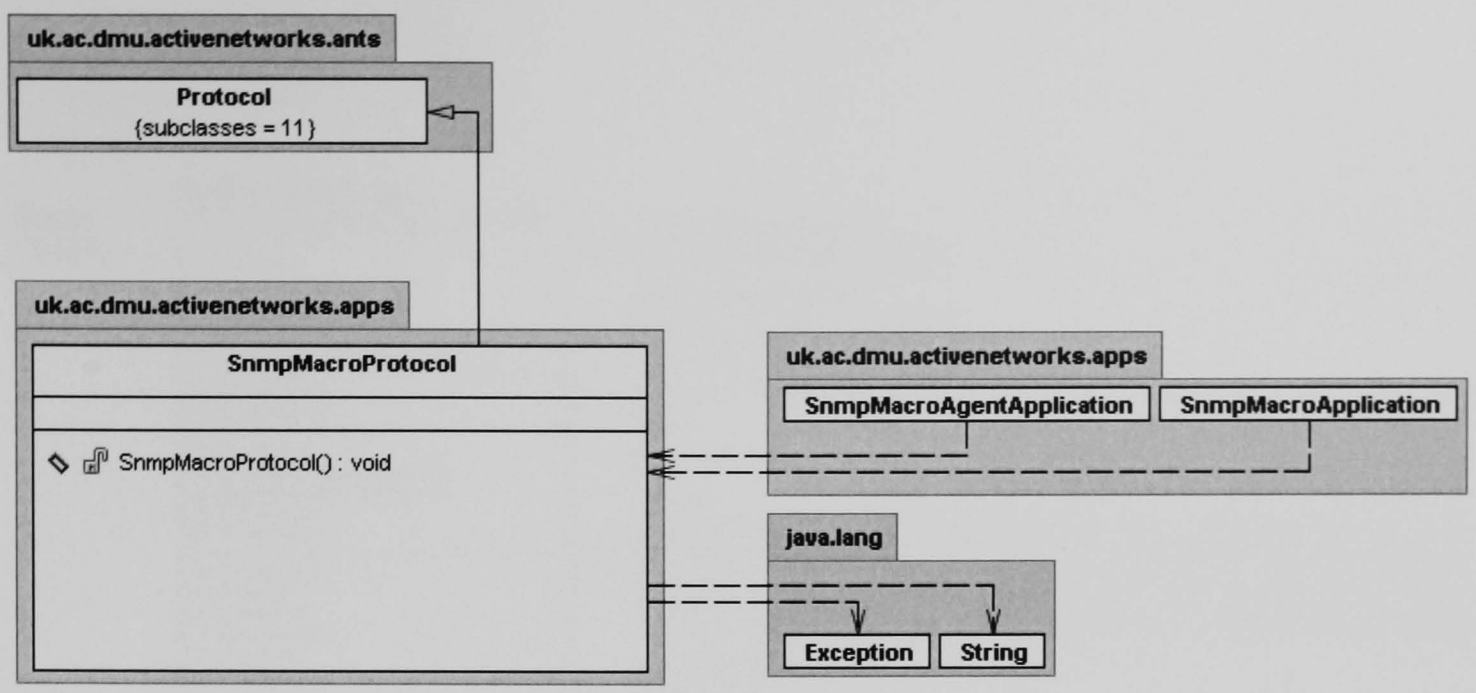
Class: SnmpMacroBean



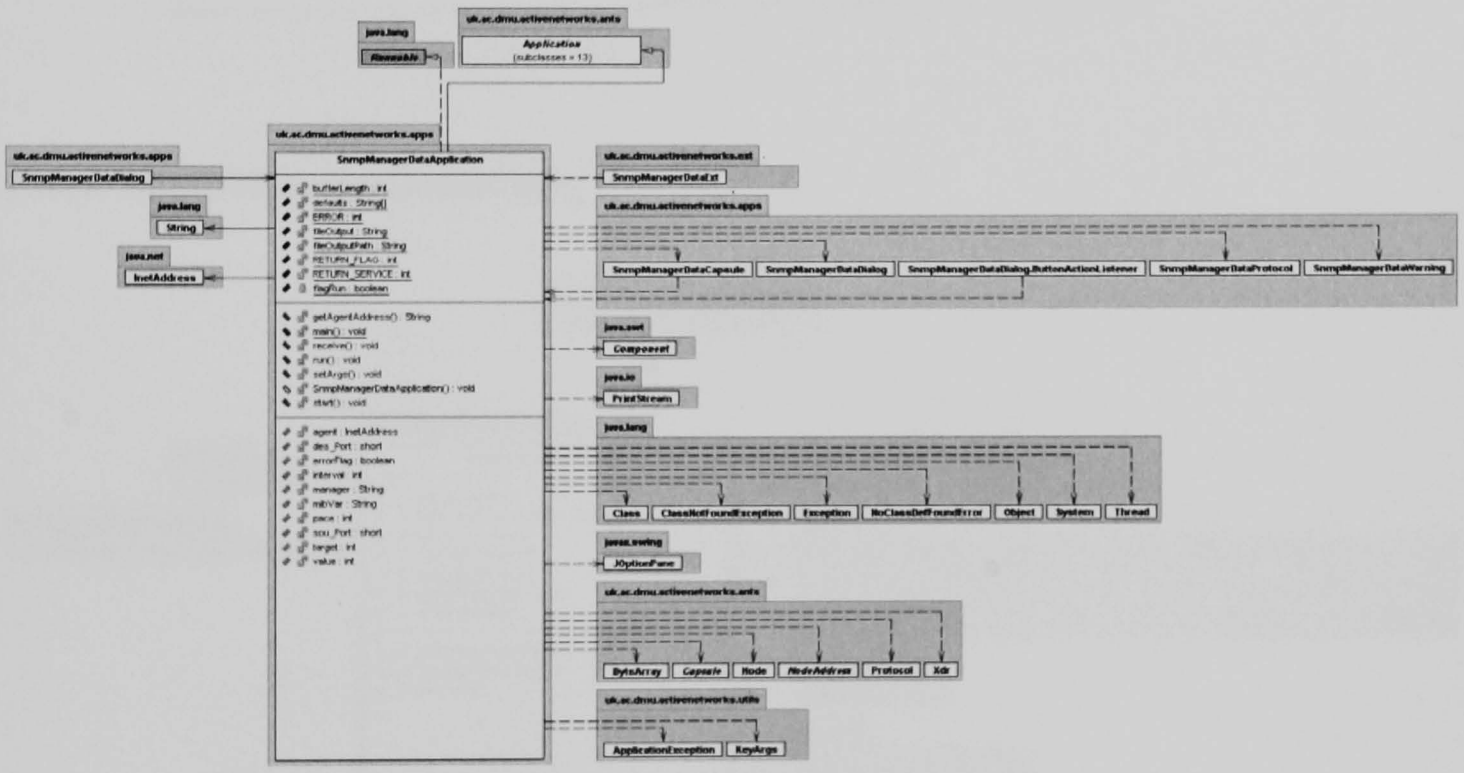
Class: SnmpMacroCapsule



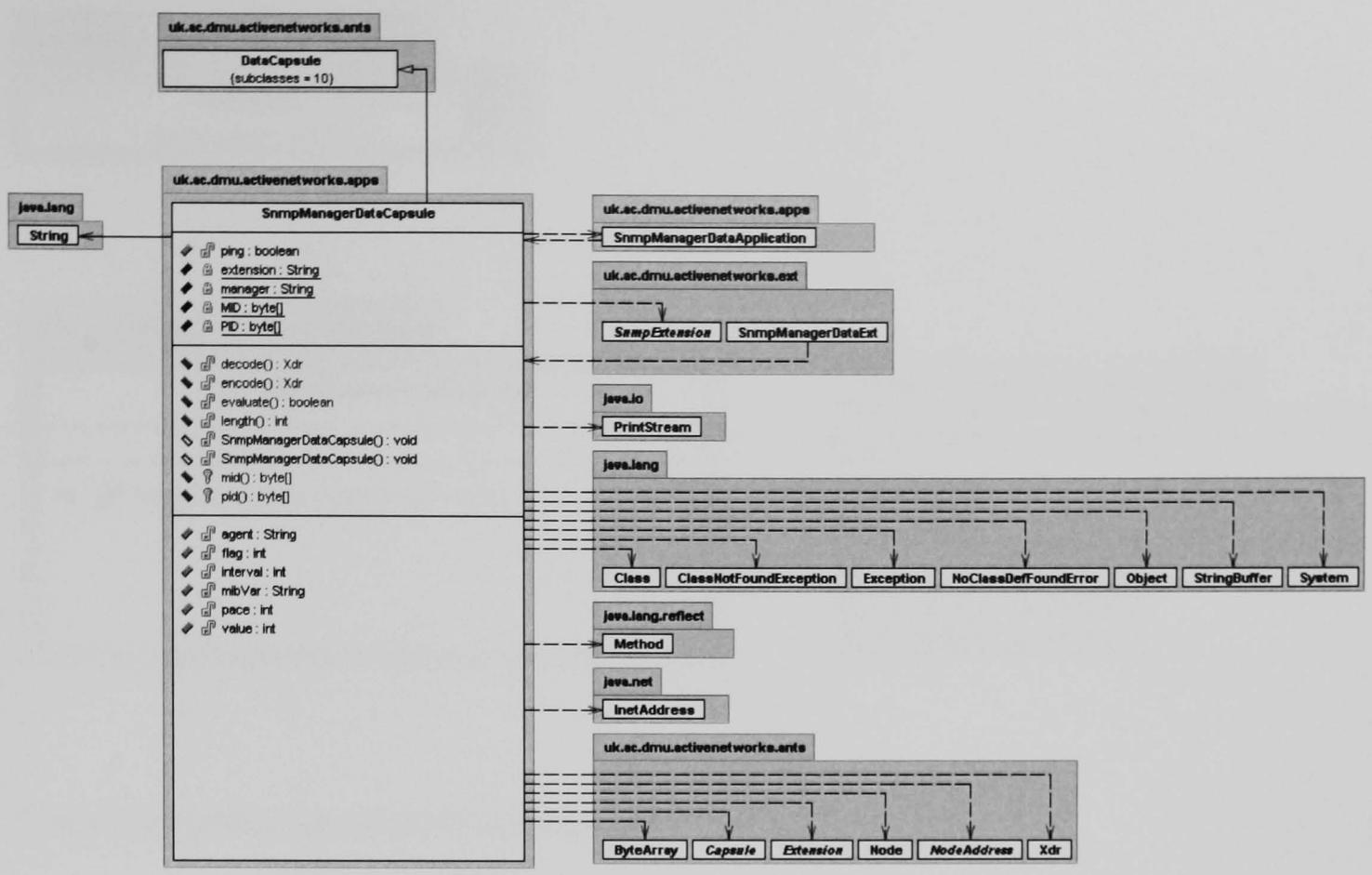
Class: SnmpMacroProtocol



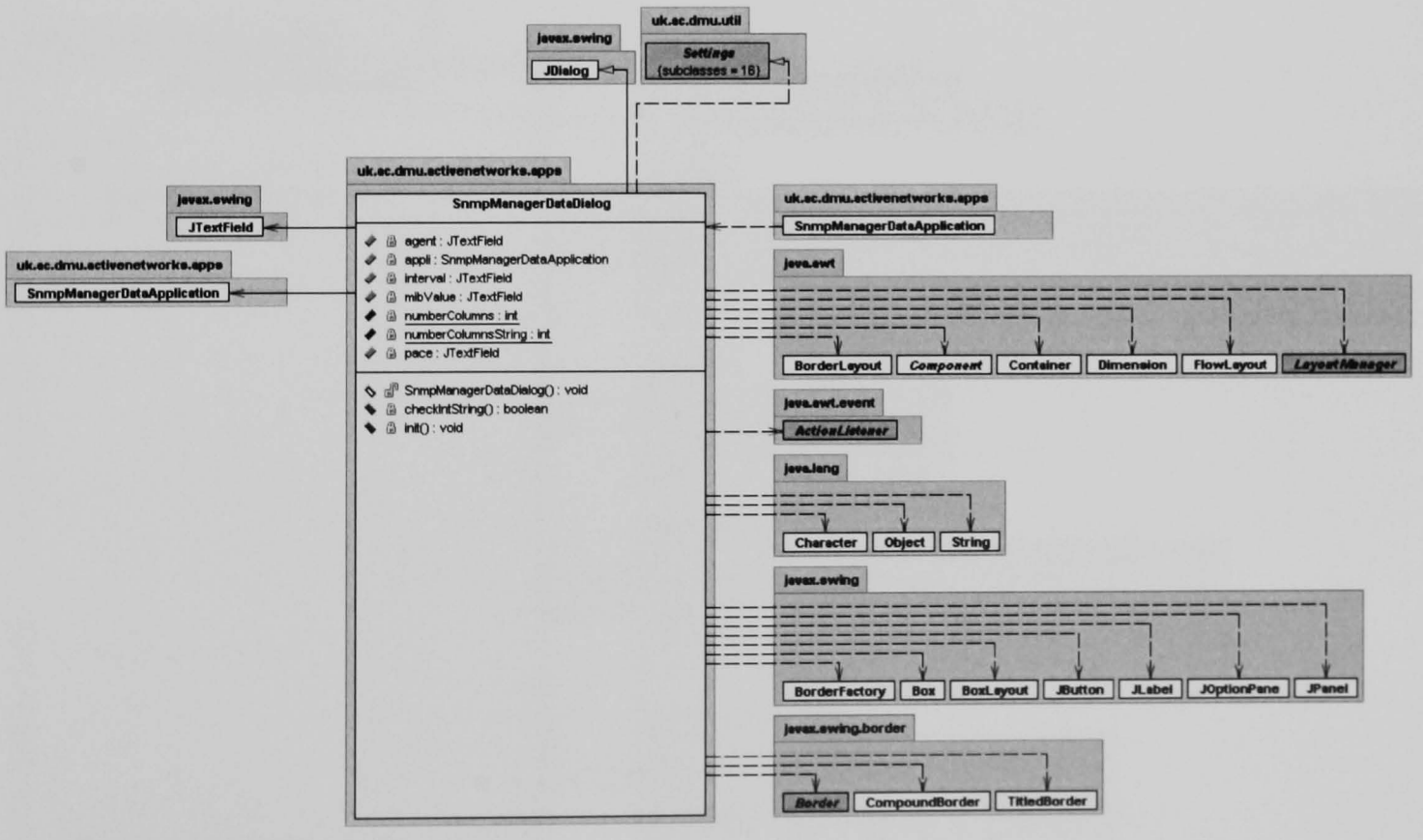
Class: SnmpManagerDataApplication



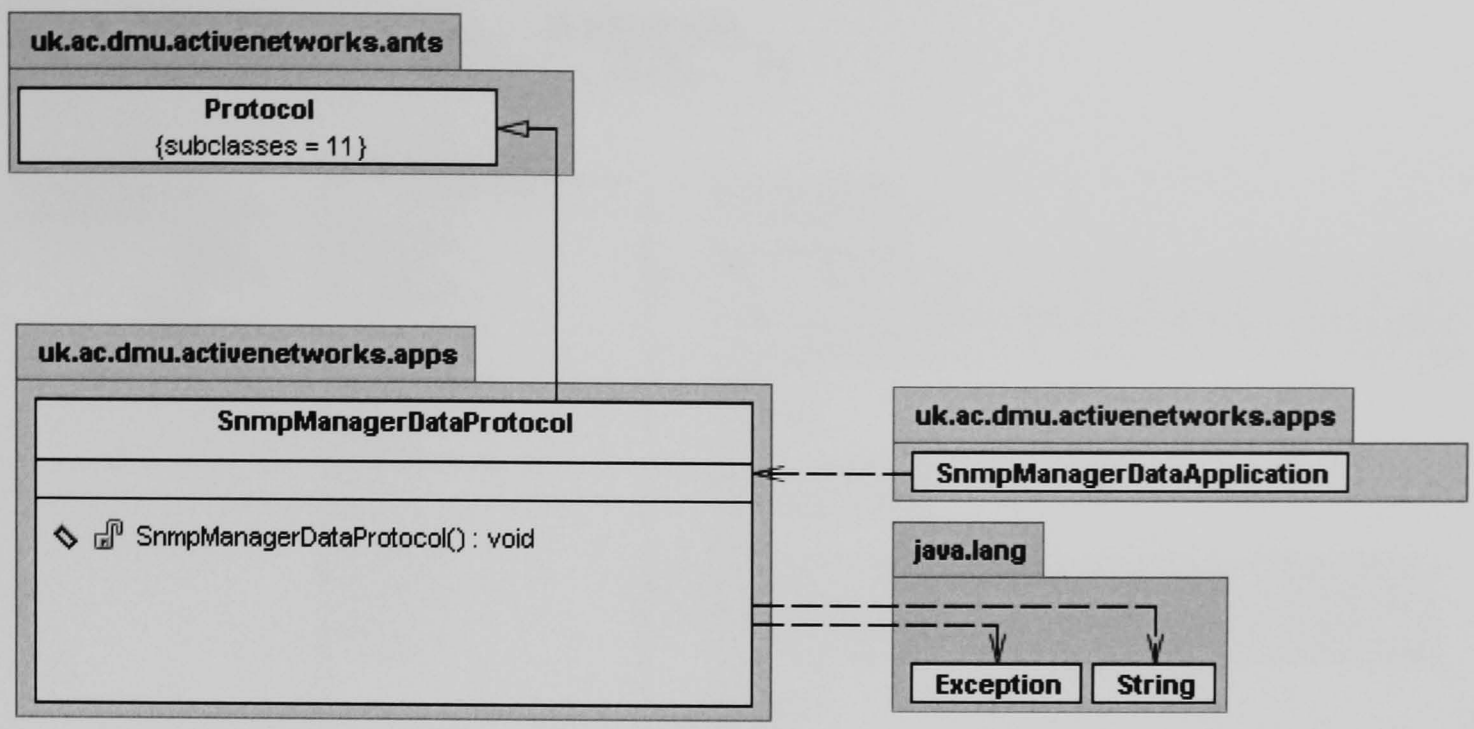
Class: SnmpManagerDataCapsule



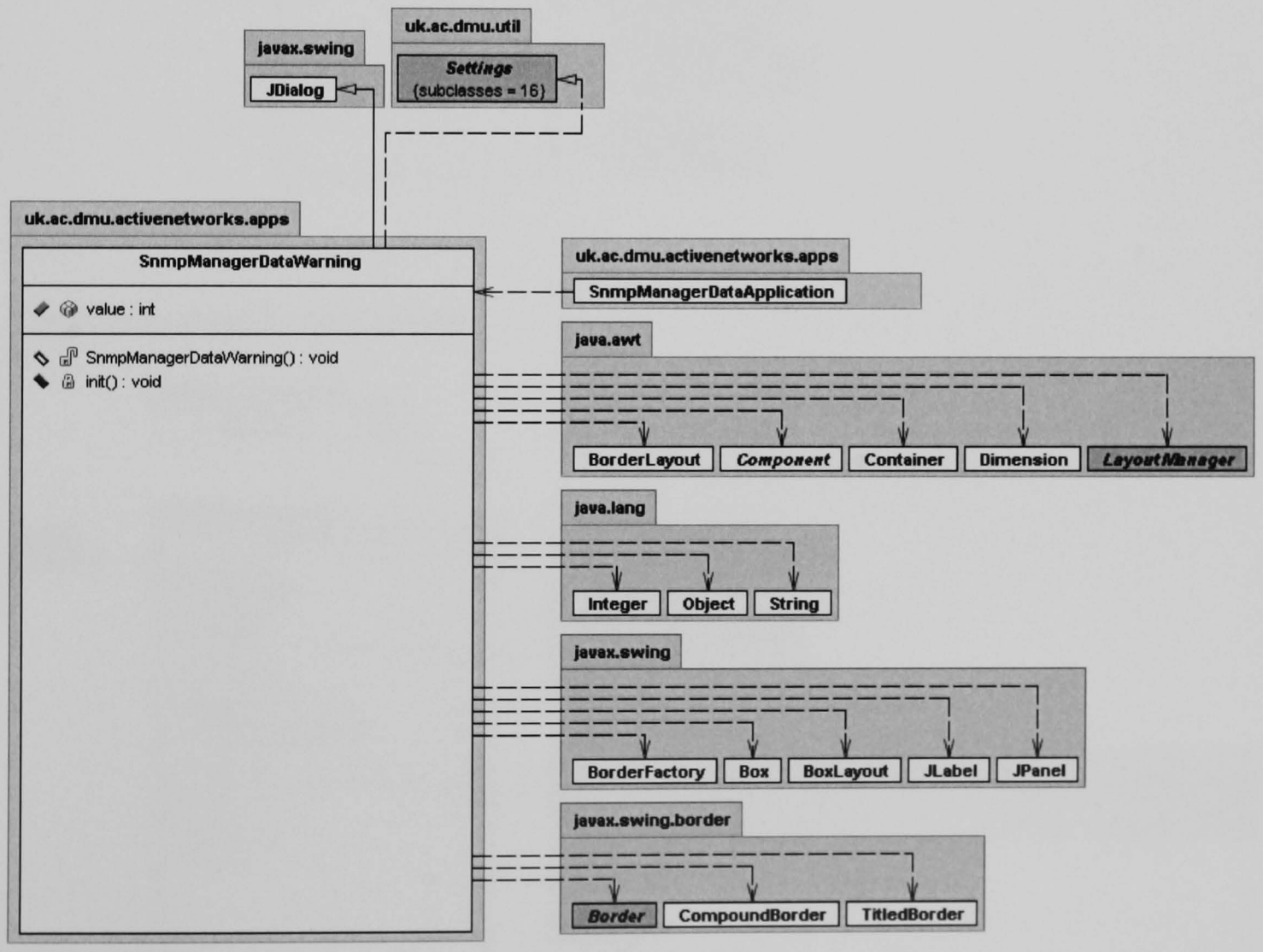
Class: SnmpManagerDataDialog



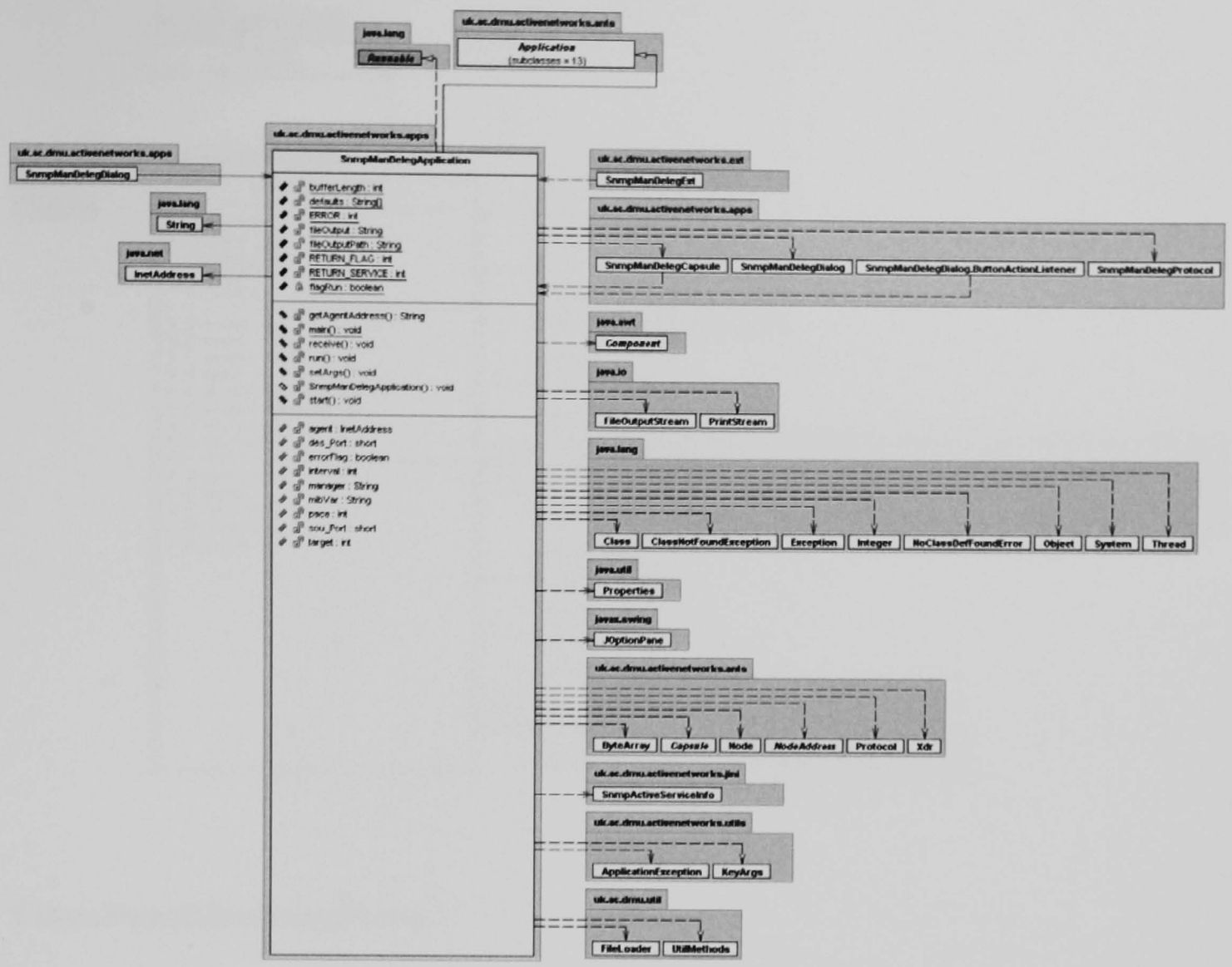
Class: SnmpManagerDataProtocol



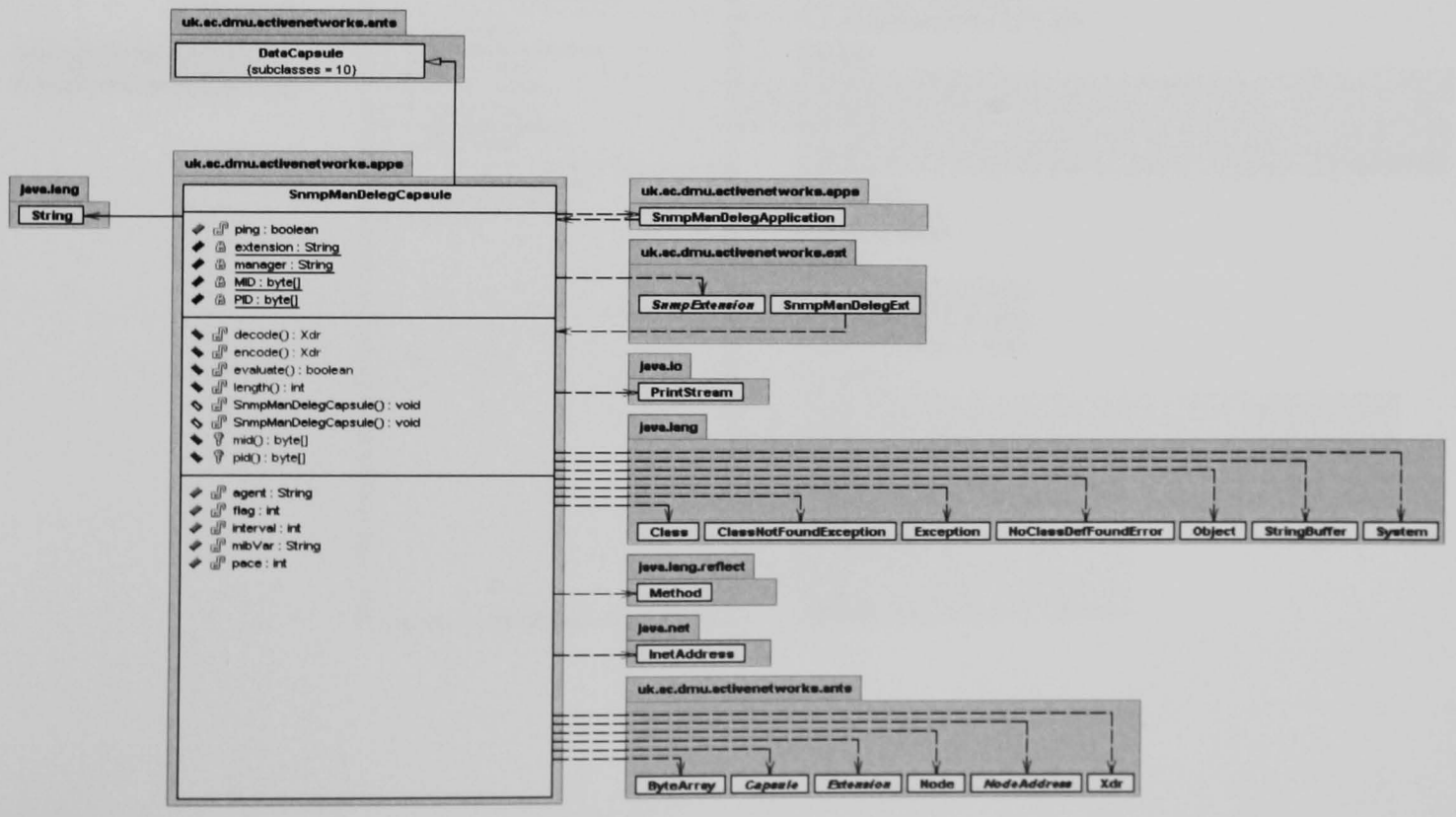
Class:SnmpManagerDataWarning



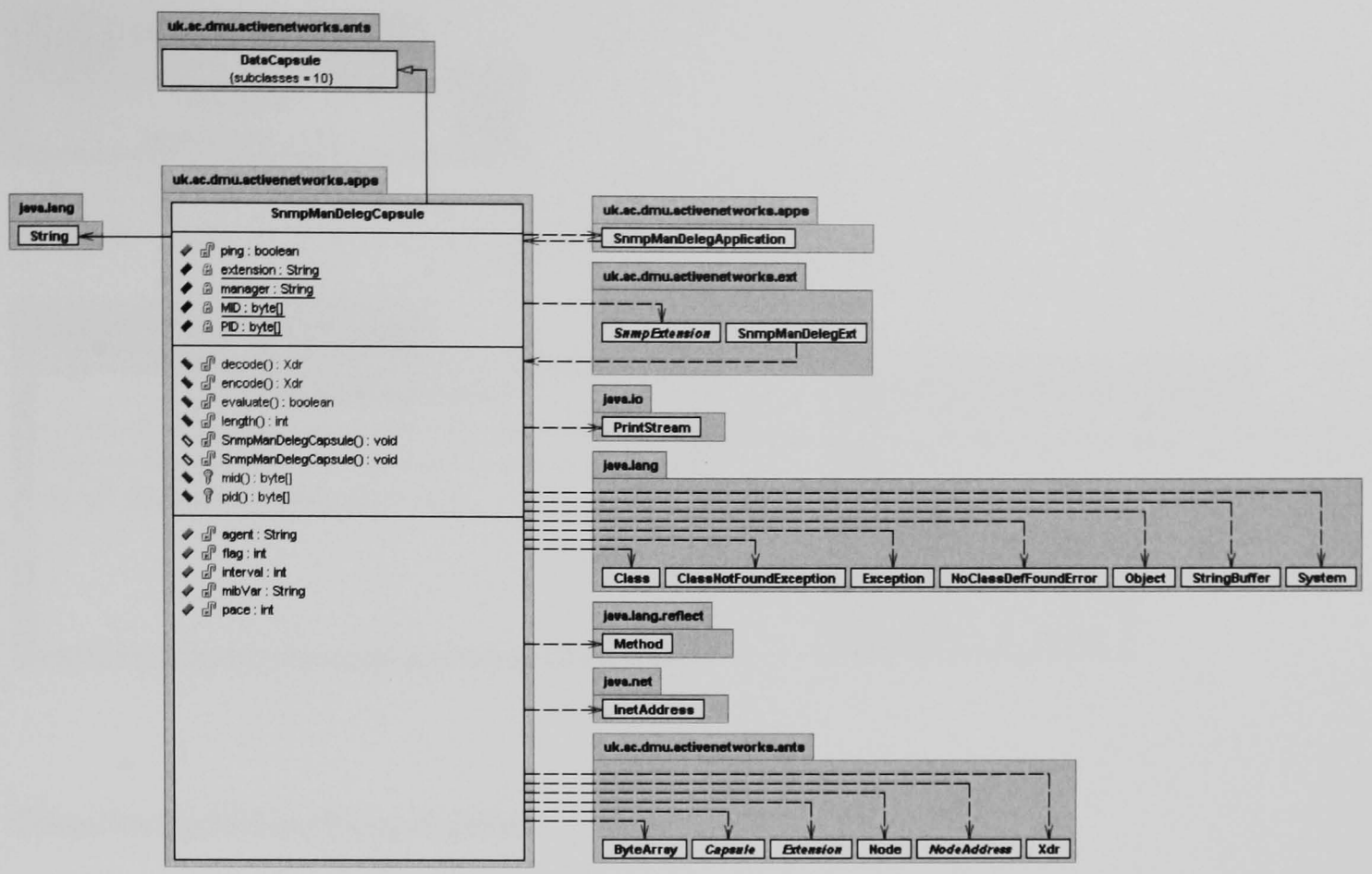
Class: SnmpManDelegApplication



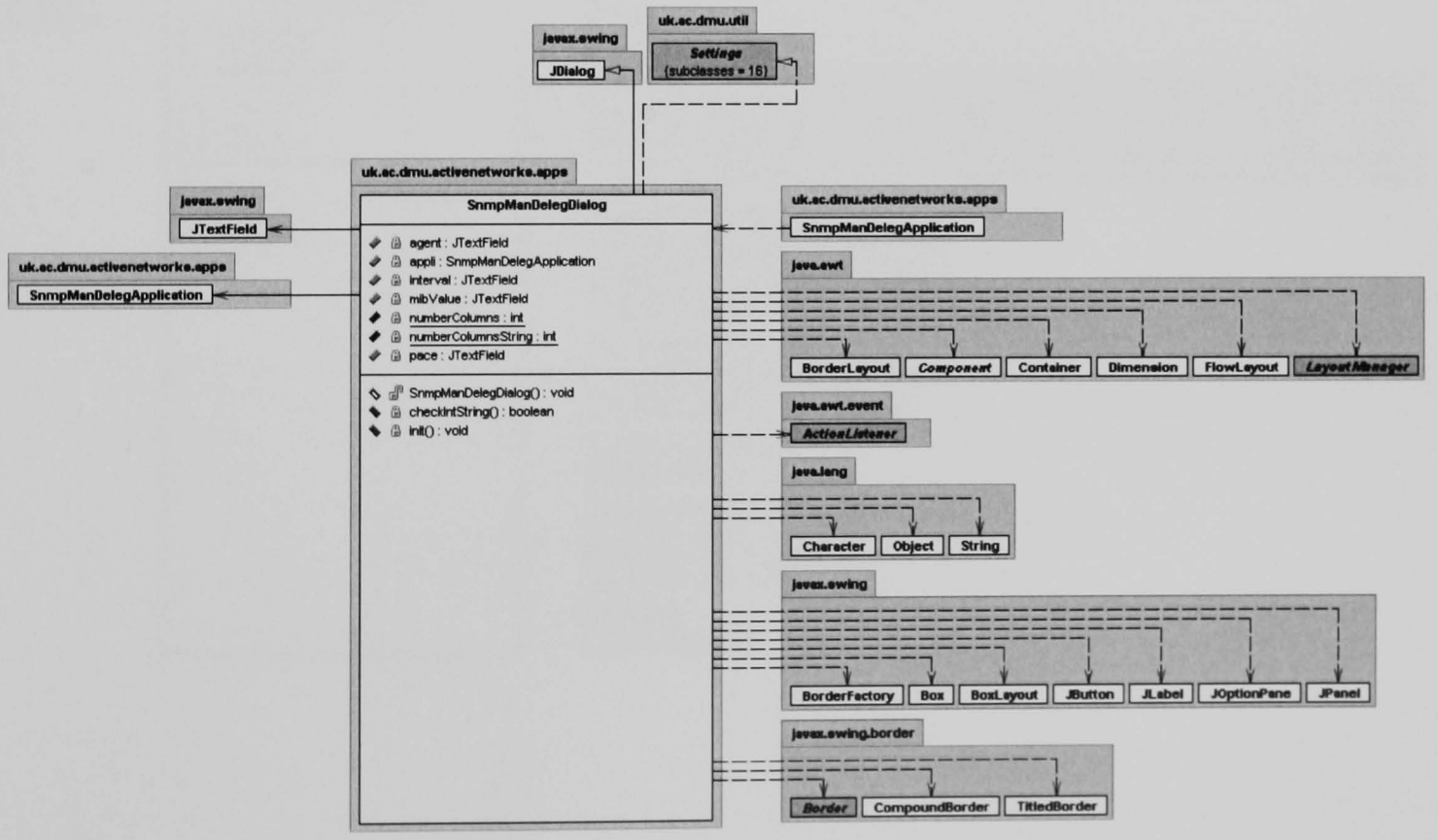
Class SnmpManDelegCapsule



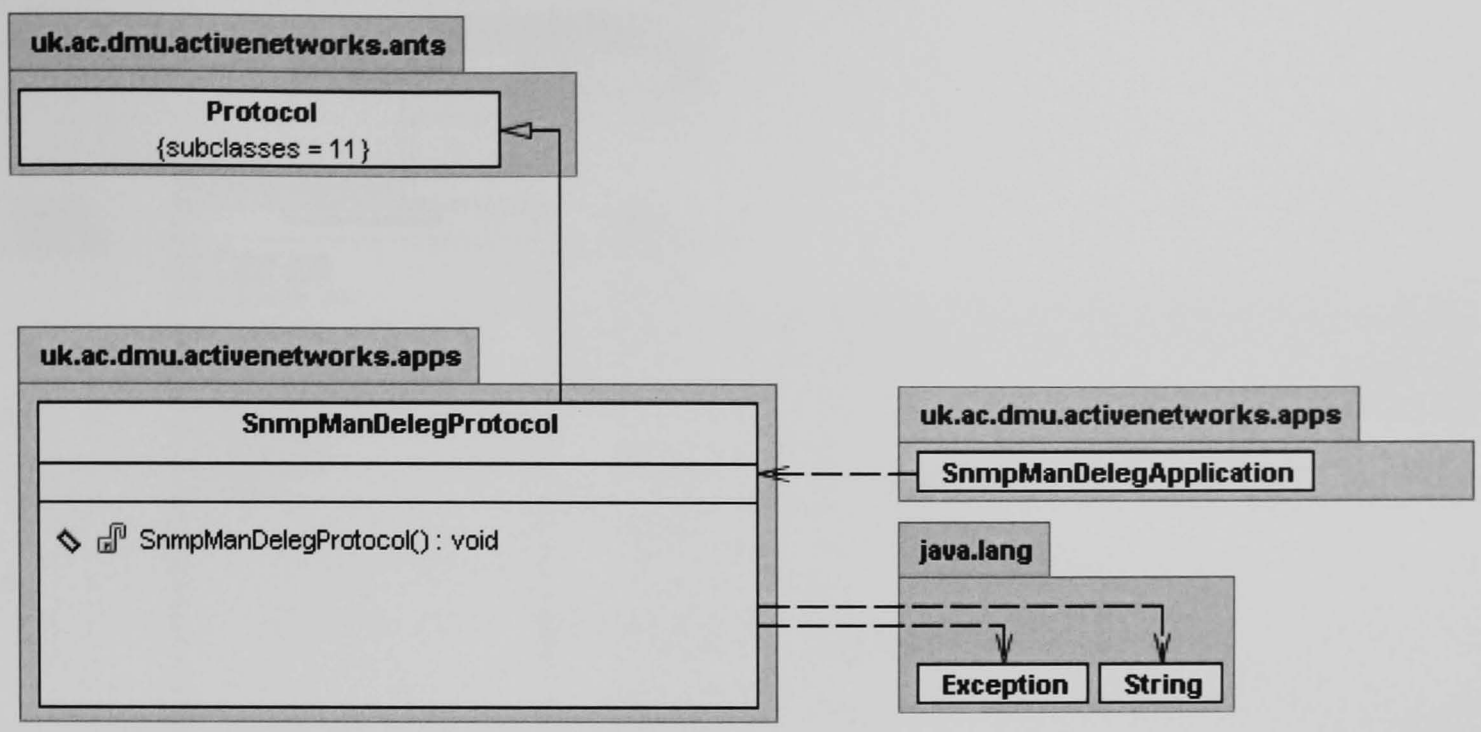
Class: SnmpManDelegCapsule



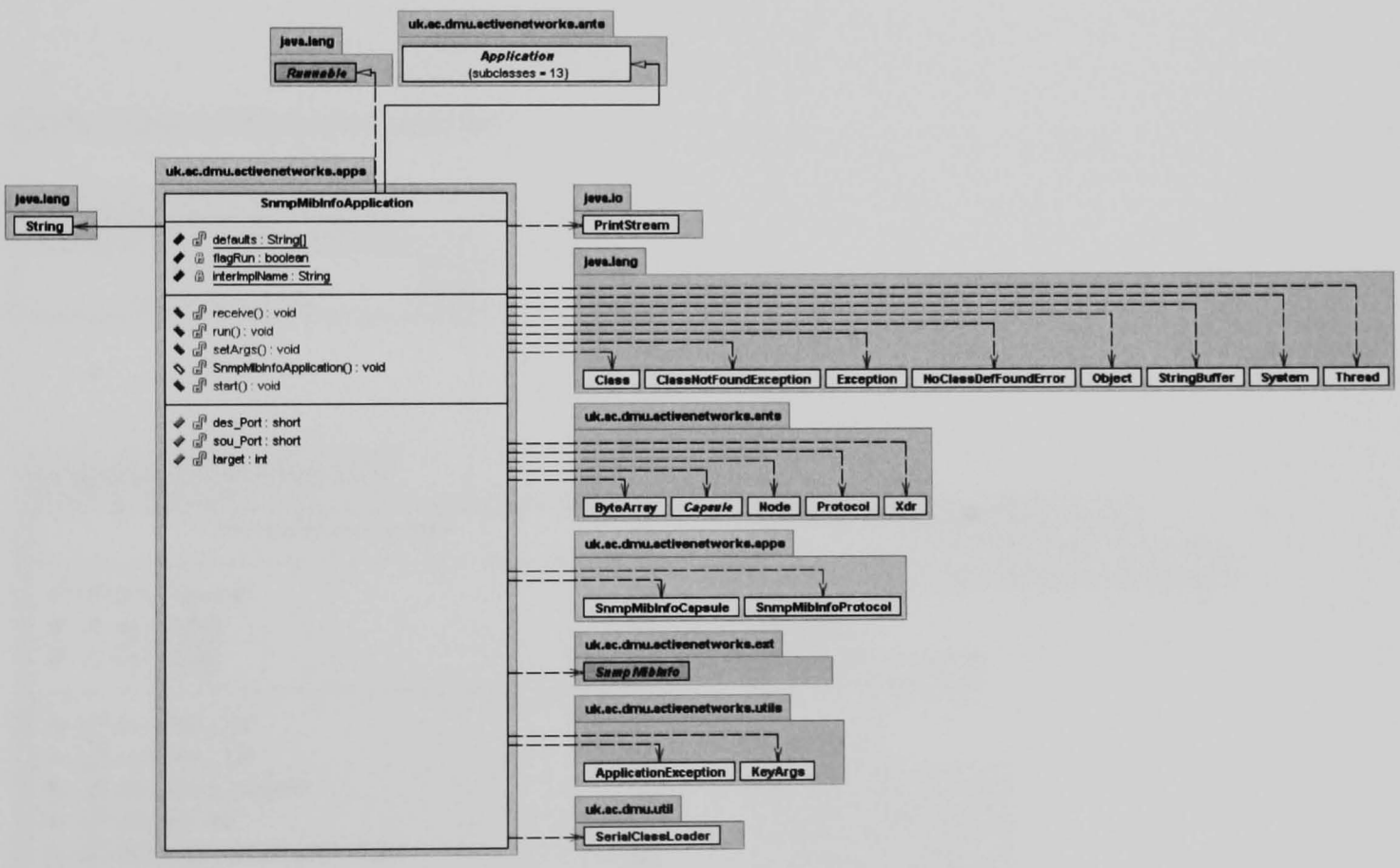
Class:SnmpManDelegDialog



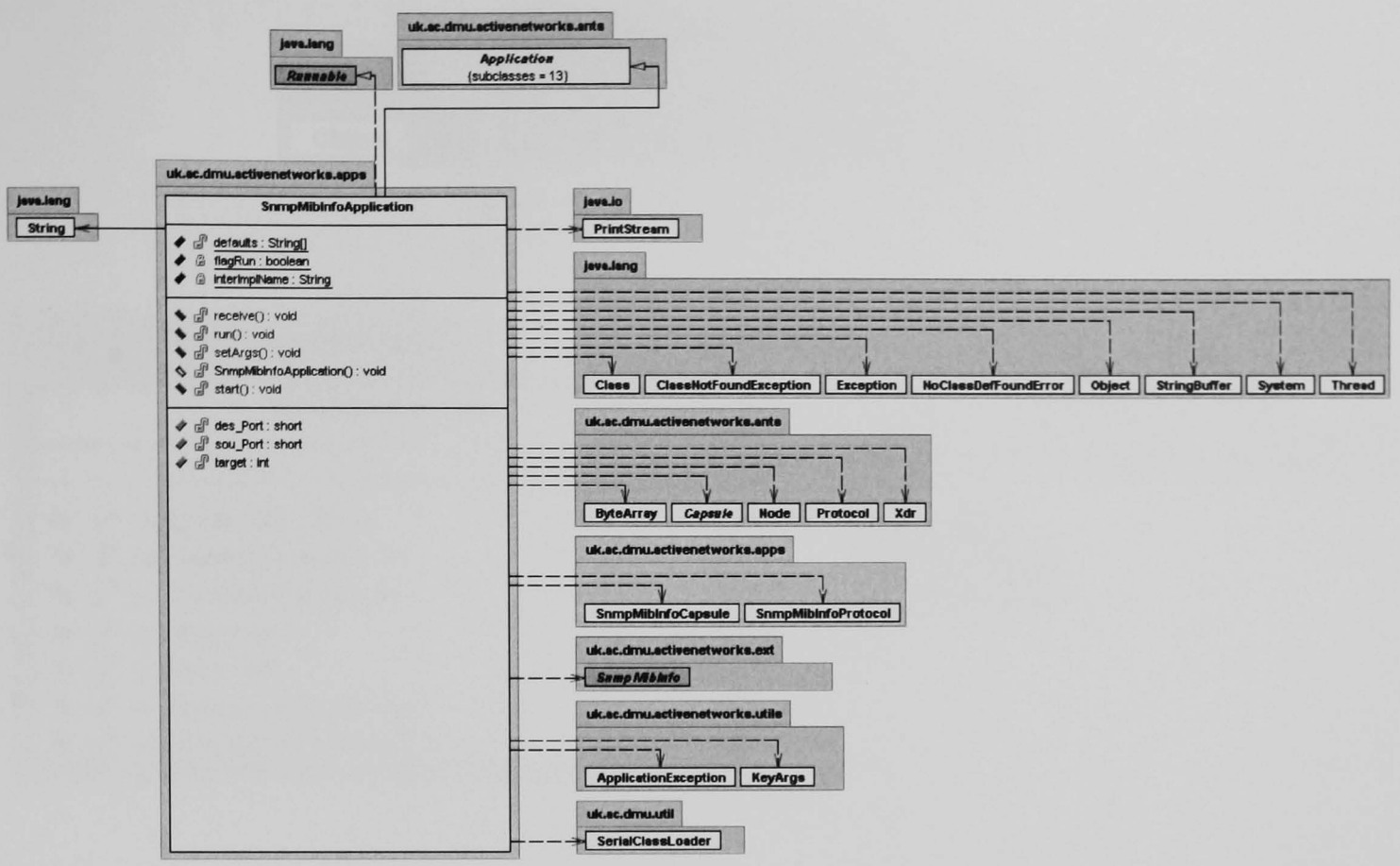
Class: SnmpManDelegProtocol



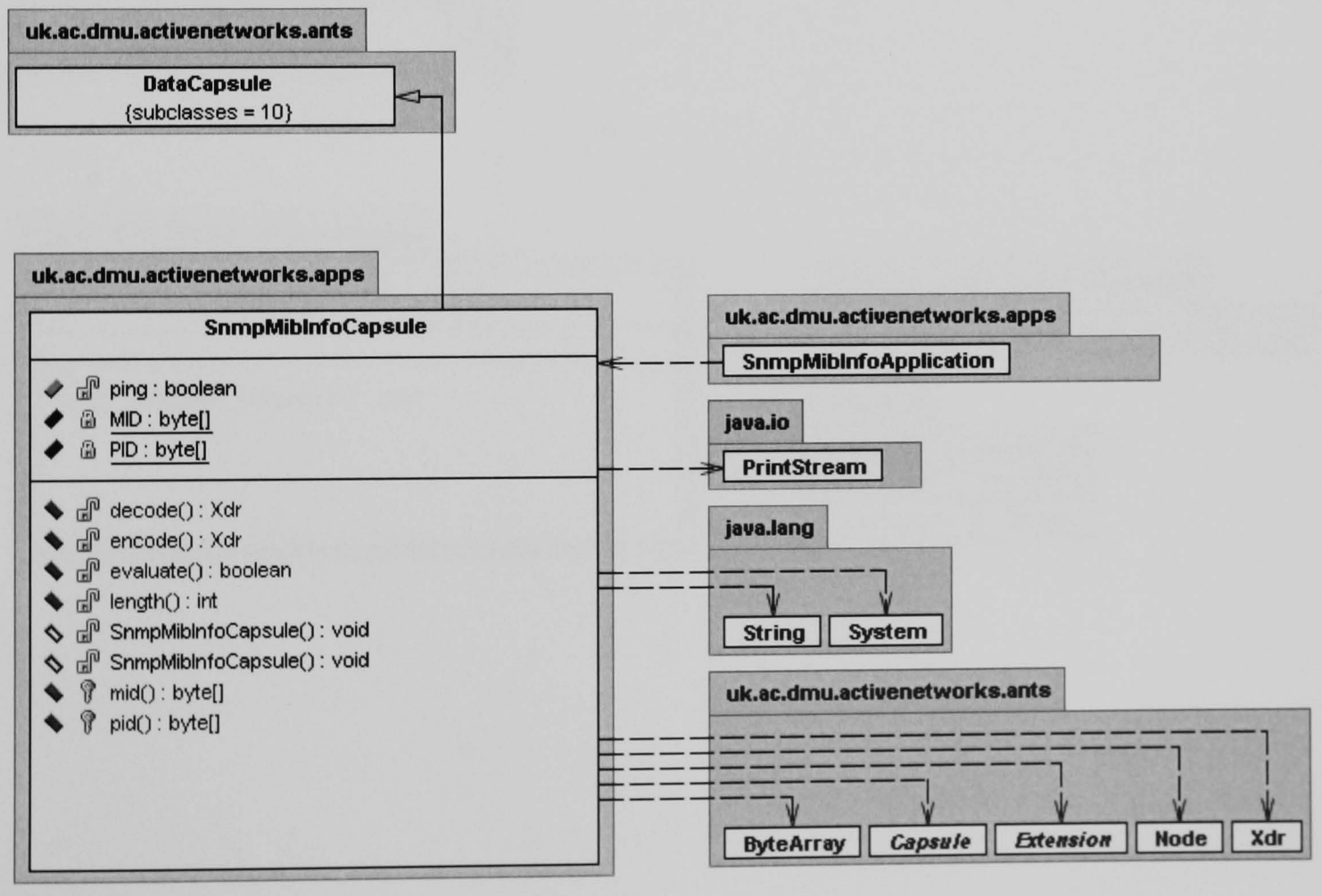
Class:SnmpMibInfoApplication



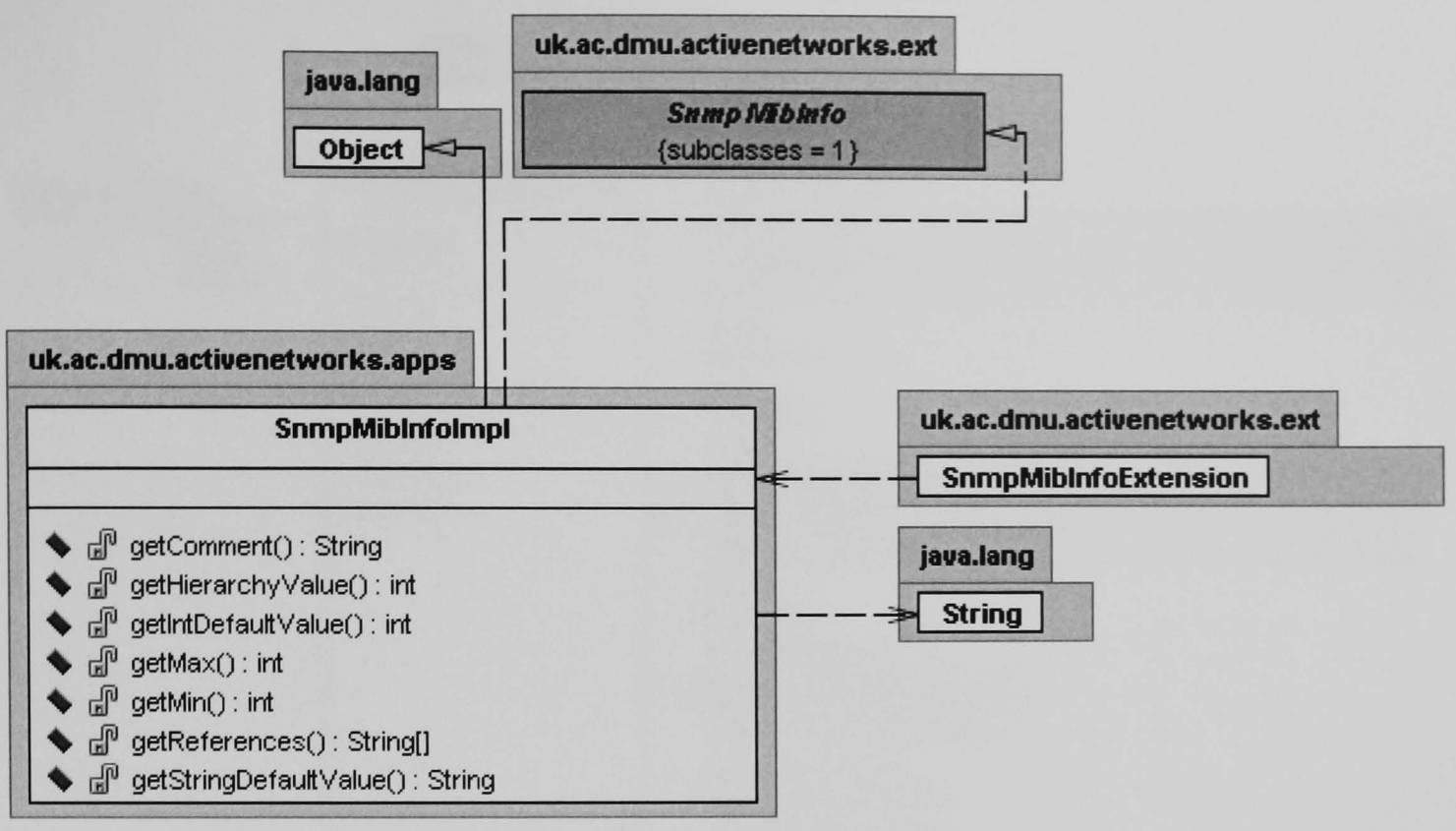
Class: SnmpMibInfoApplication



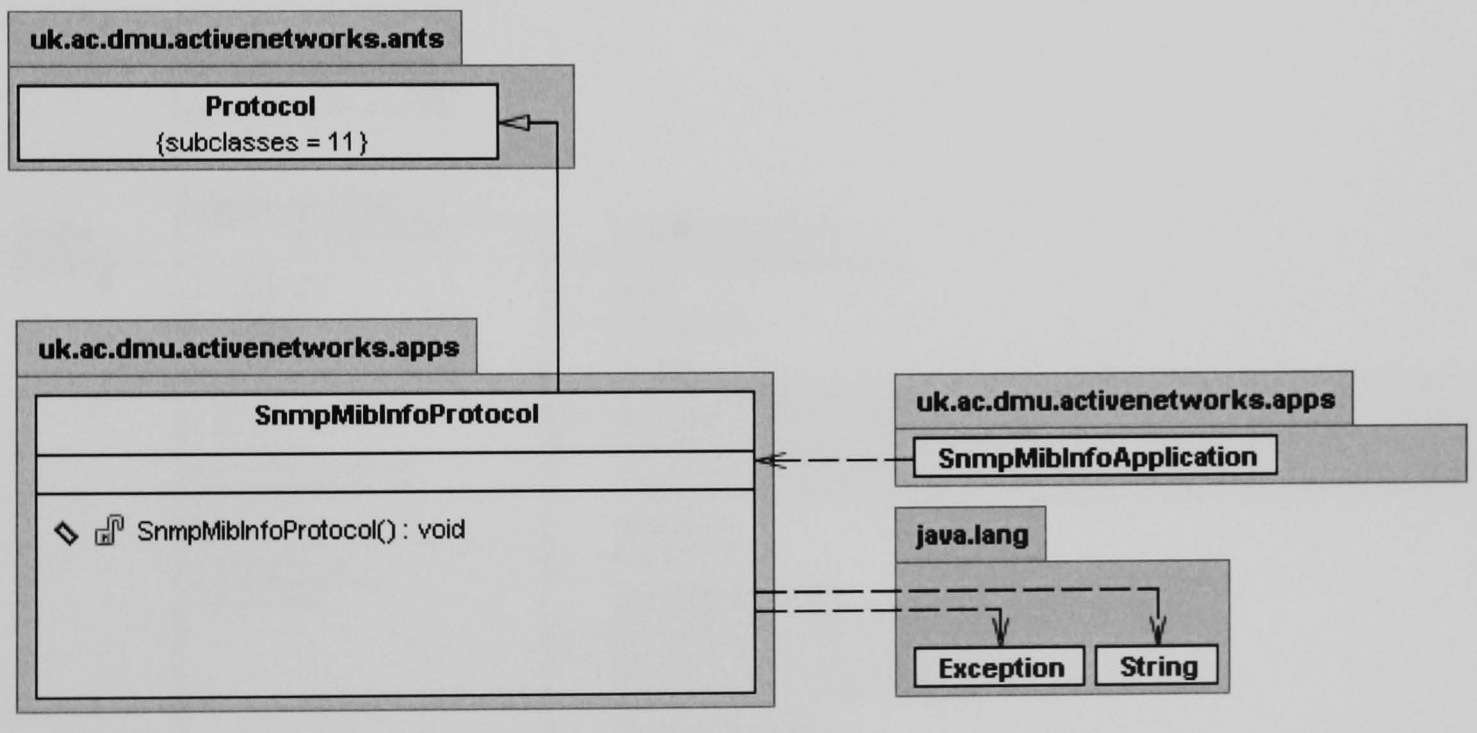
Class: SnmpMibInfoCapsule



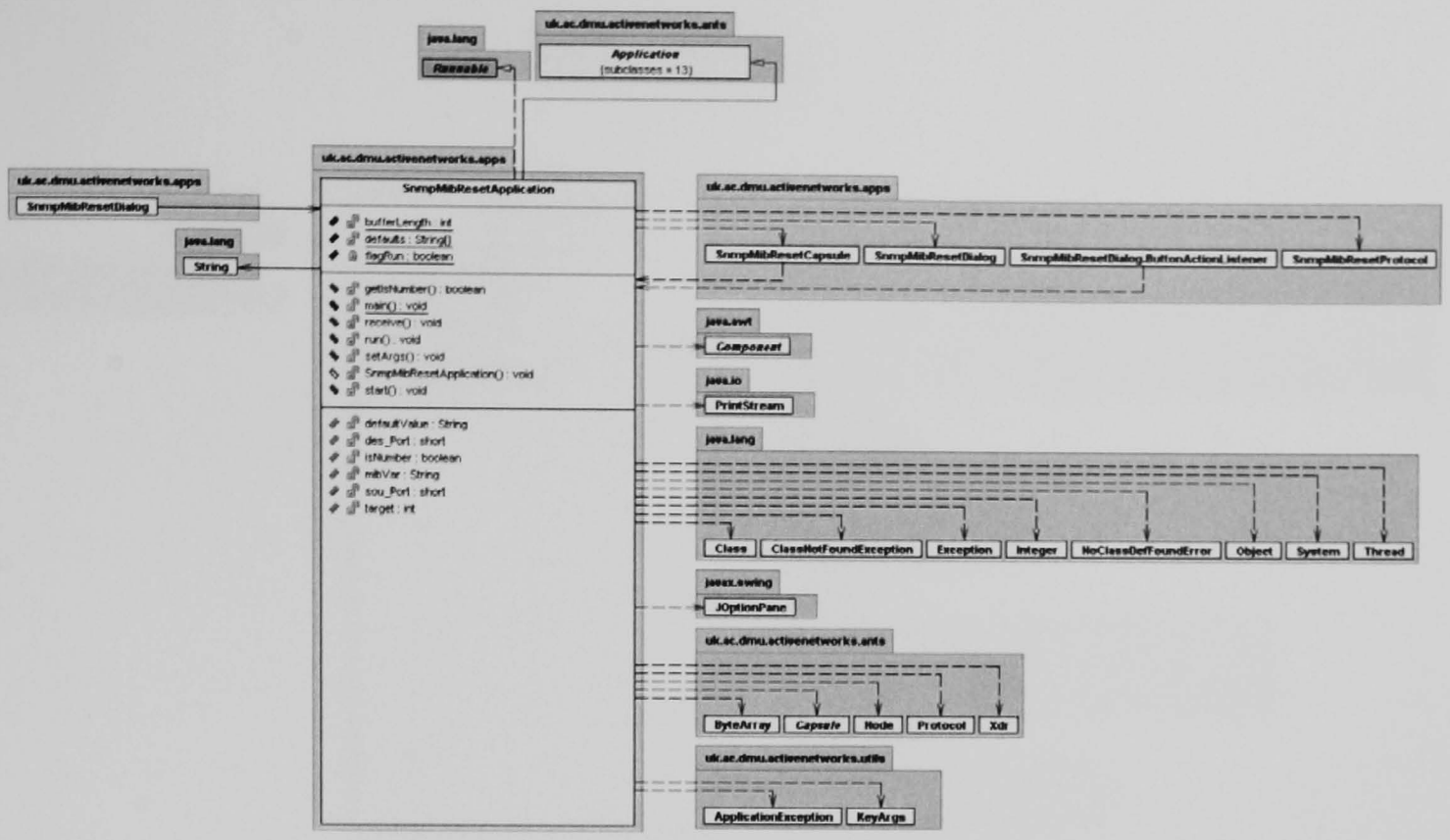
Class: SnmpMibInfoImpl



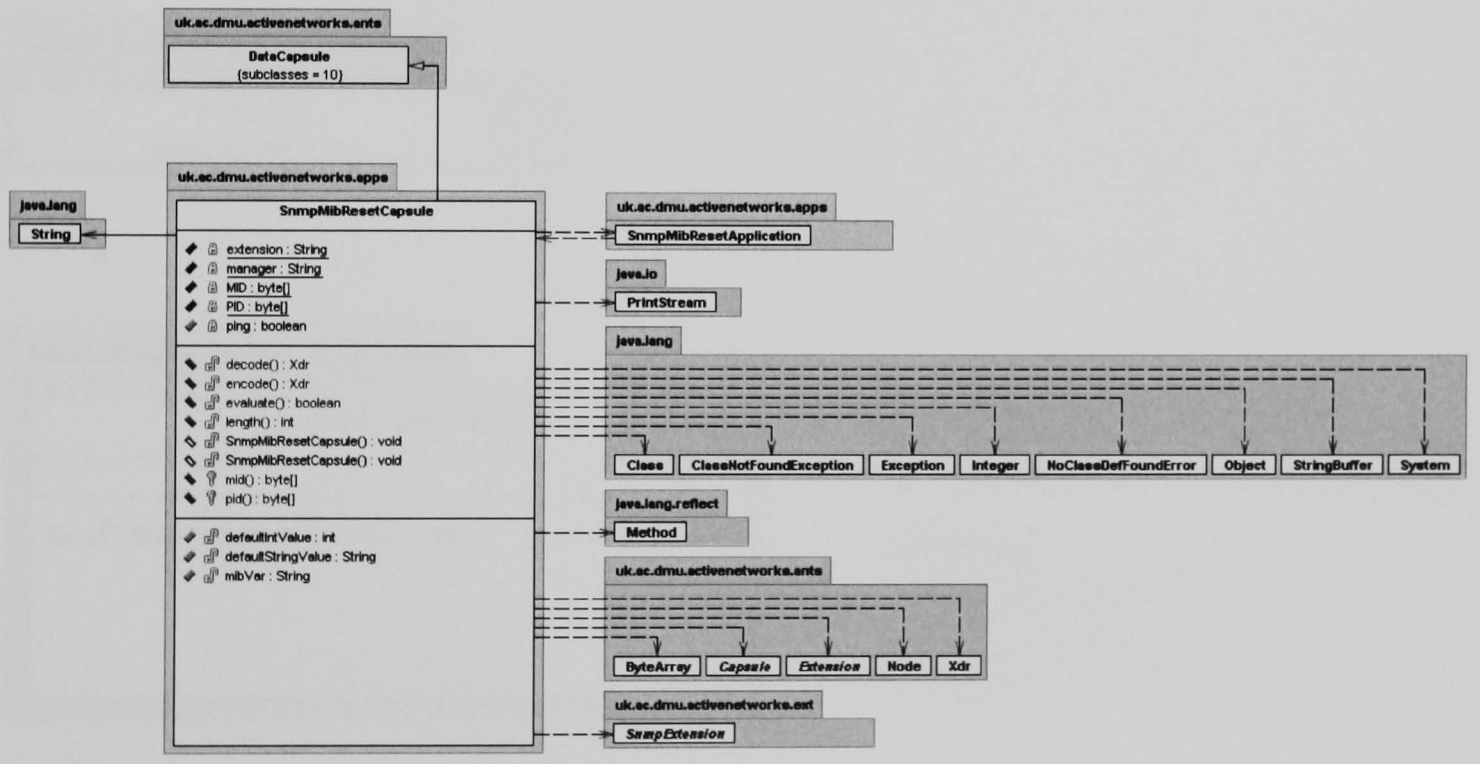
Class: SnmpMibInfoProtocol



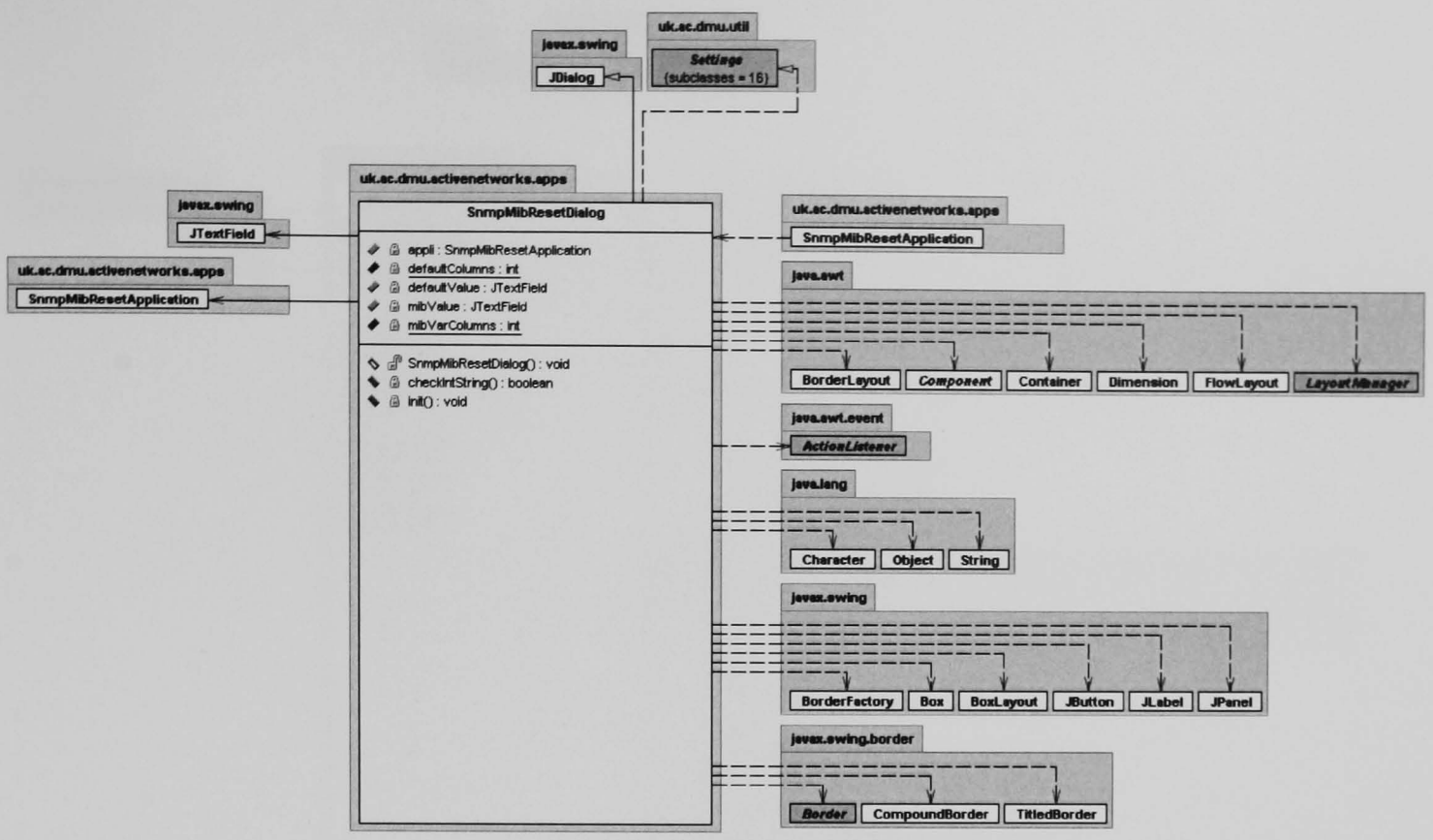
Class: SnmpMibResetApplication



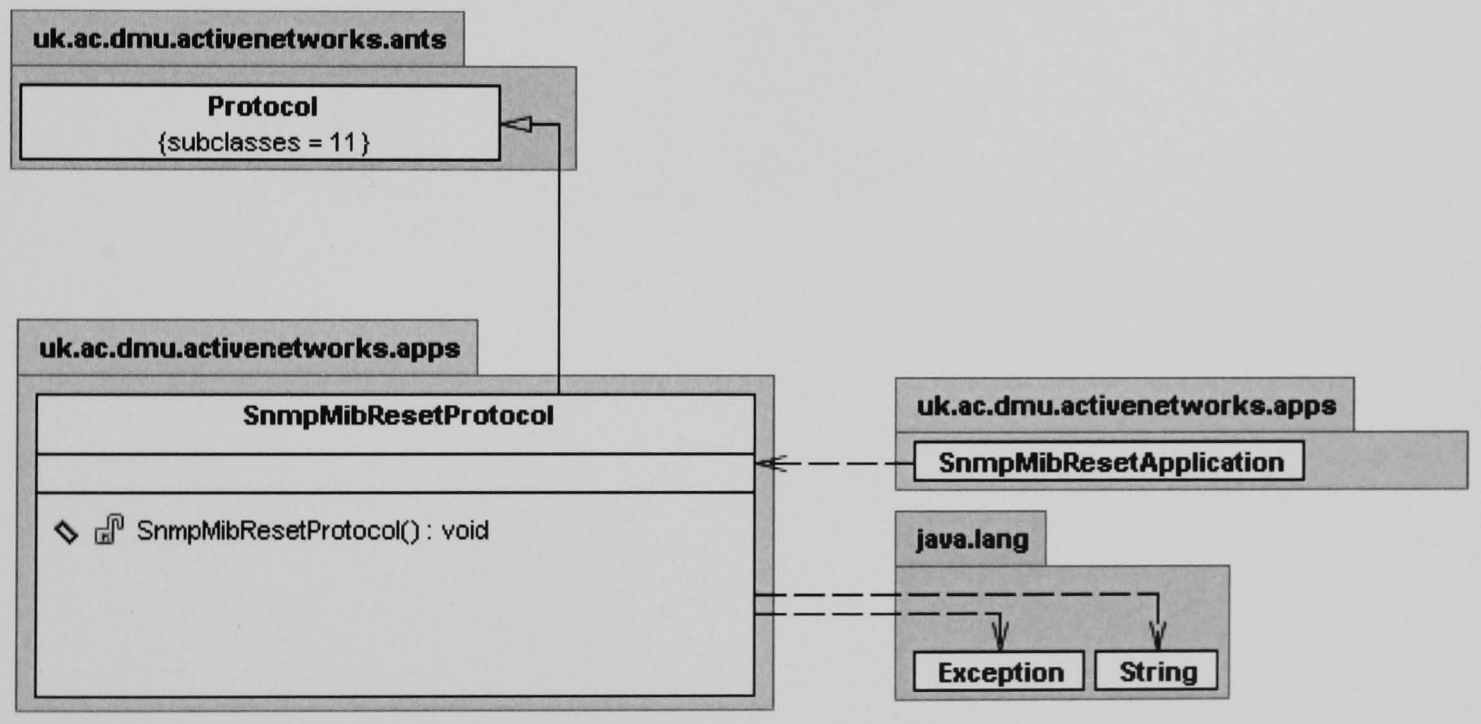
Class: SnmpMibResetCapsule



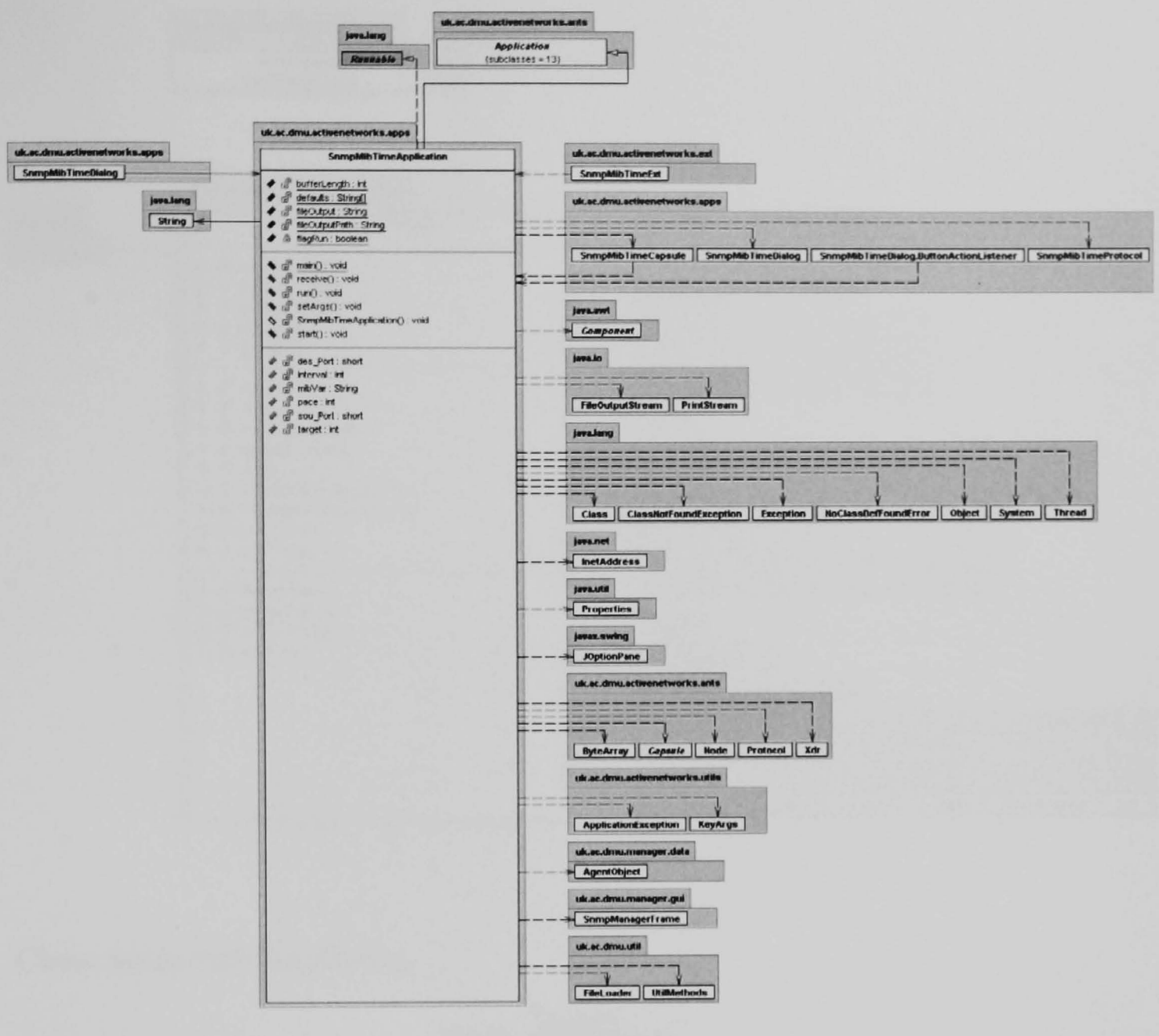
Class:SnmpMibResetDialog



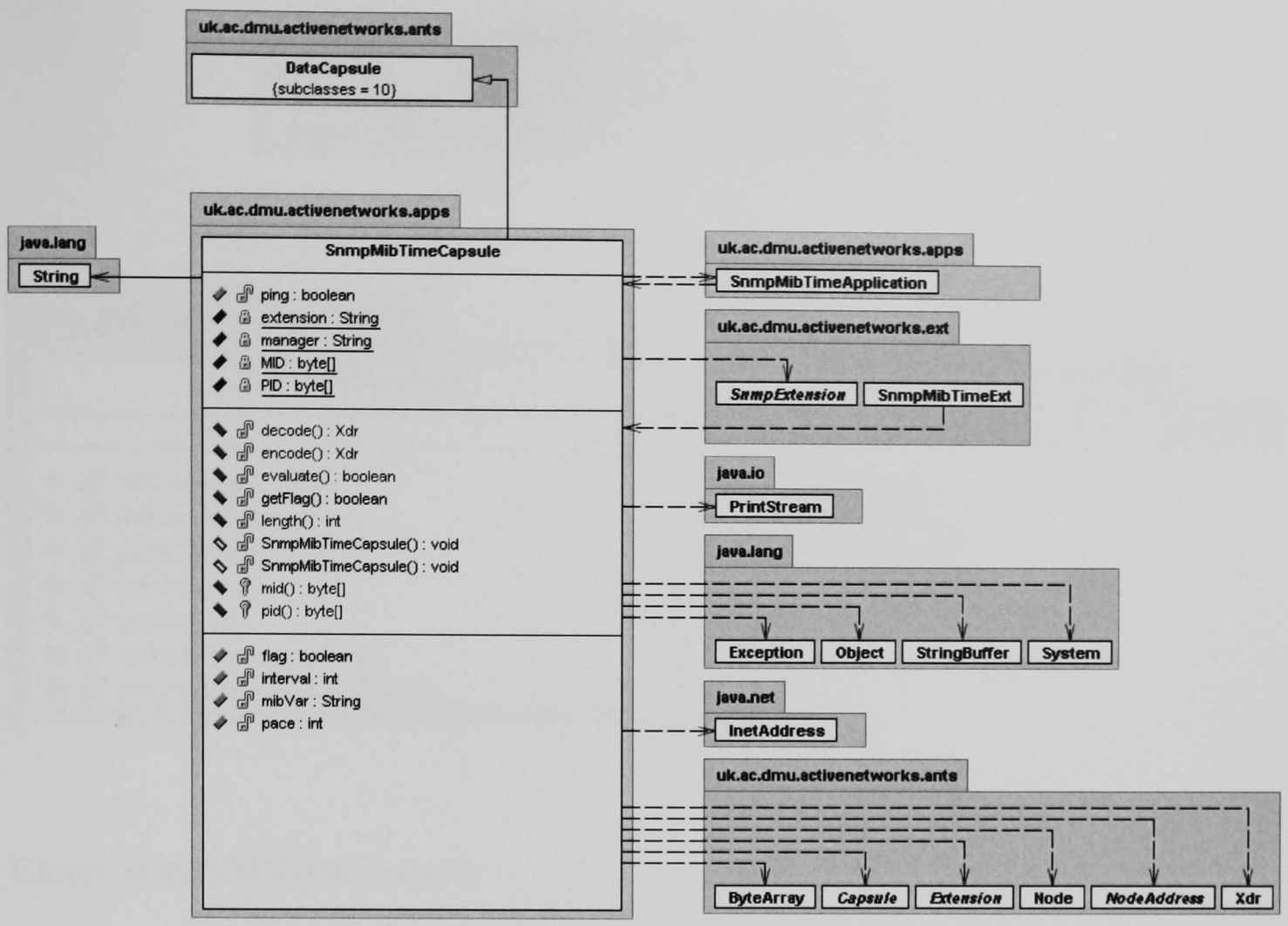
Class: SnmpMibResetProtocol



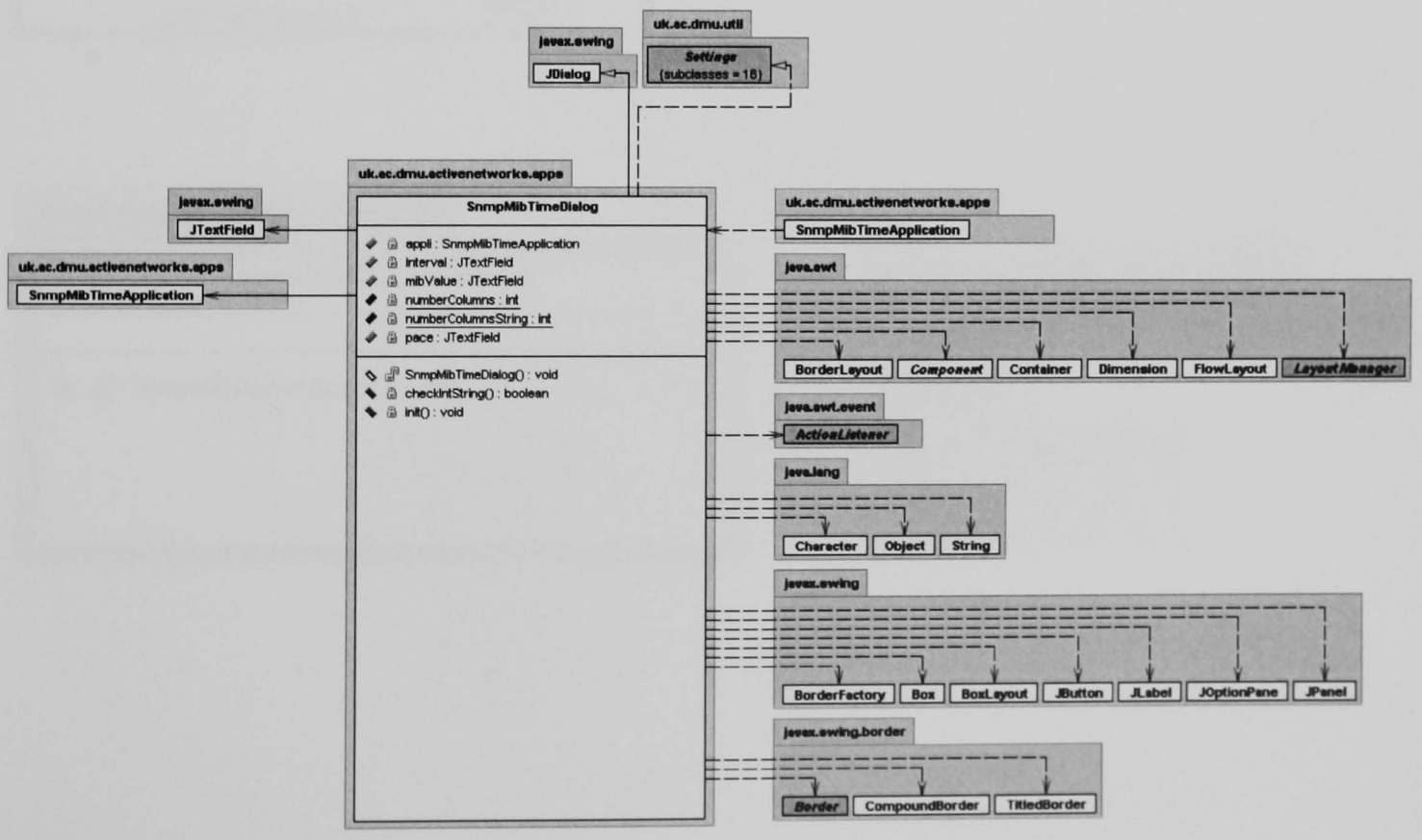
Class: SnmpMibTimeApplication



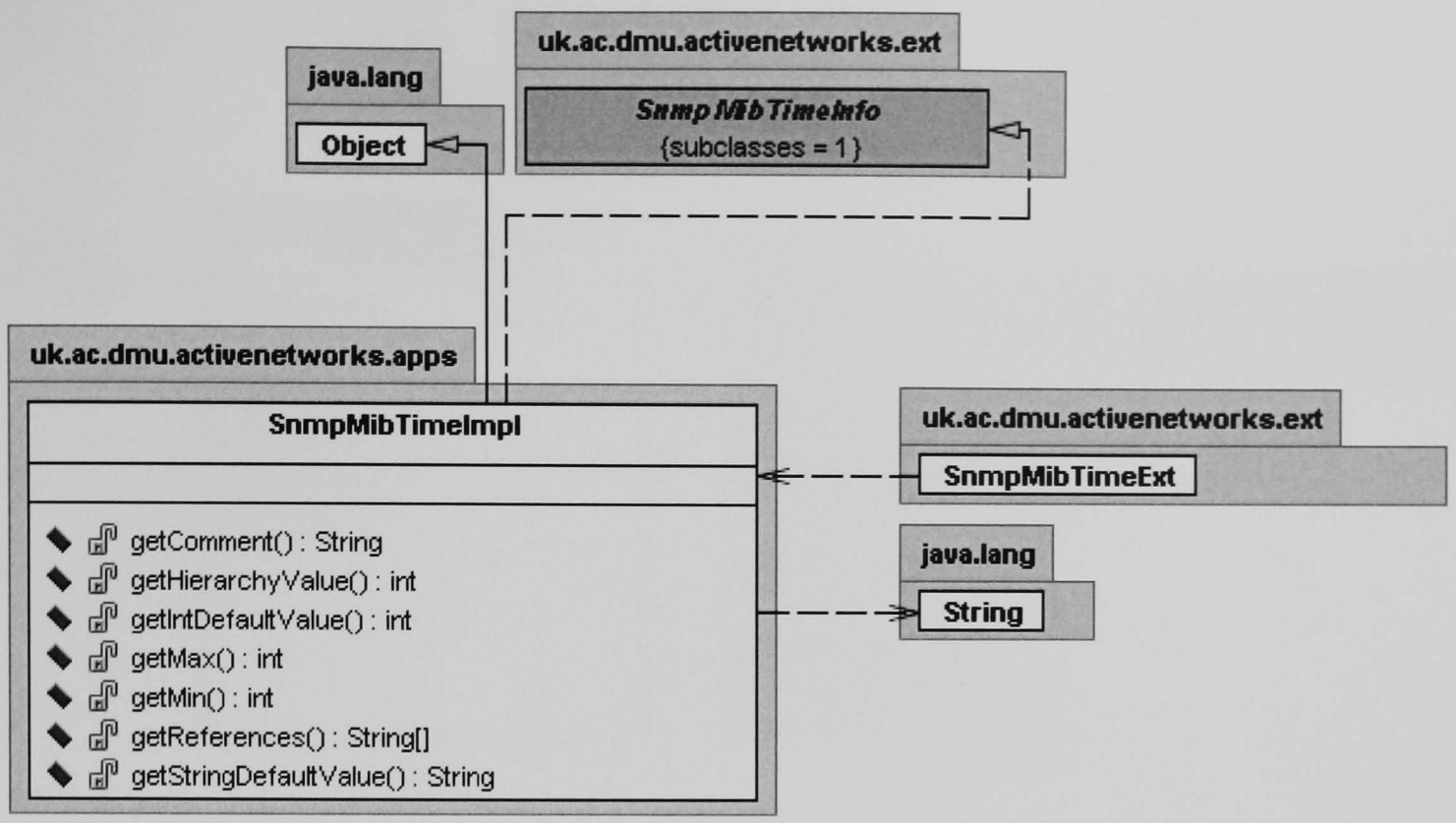
Class: SnmpMibTimeCapsule



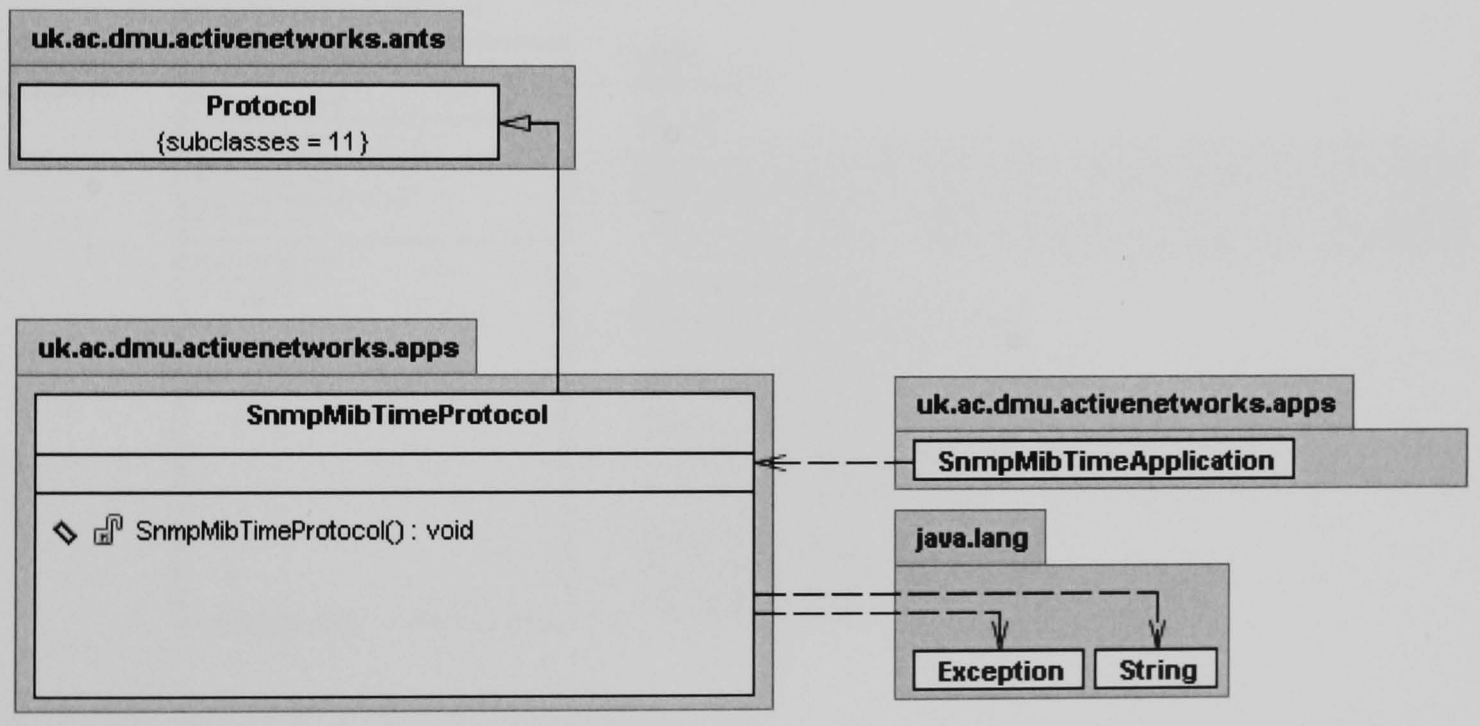
Class: SnmpMibTimeDialog



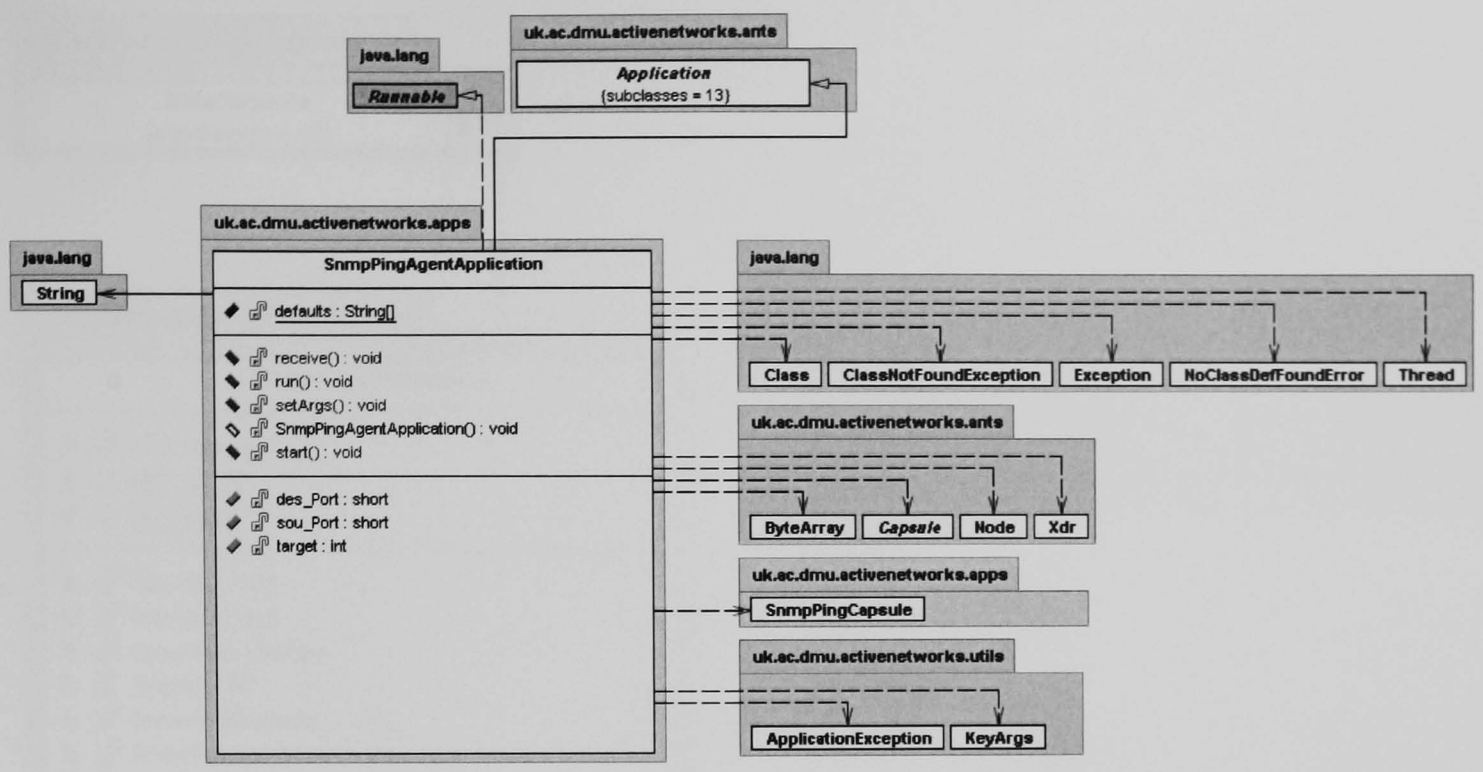
Class: SnmpMibTimeImpl



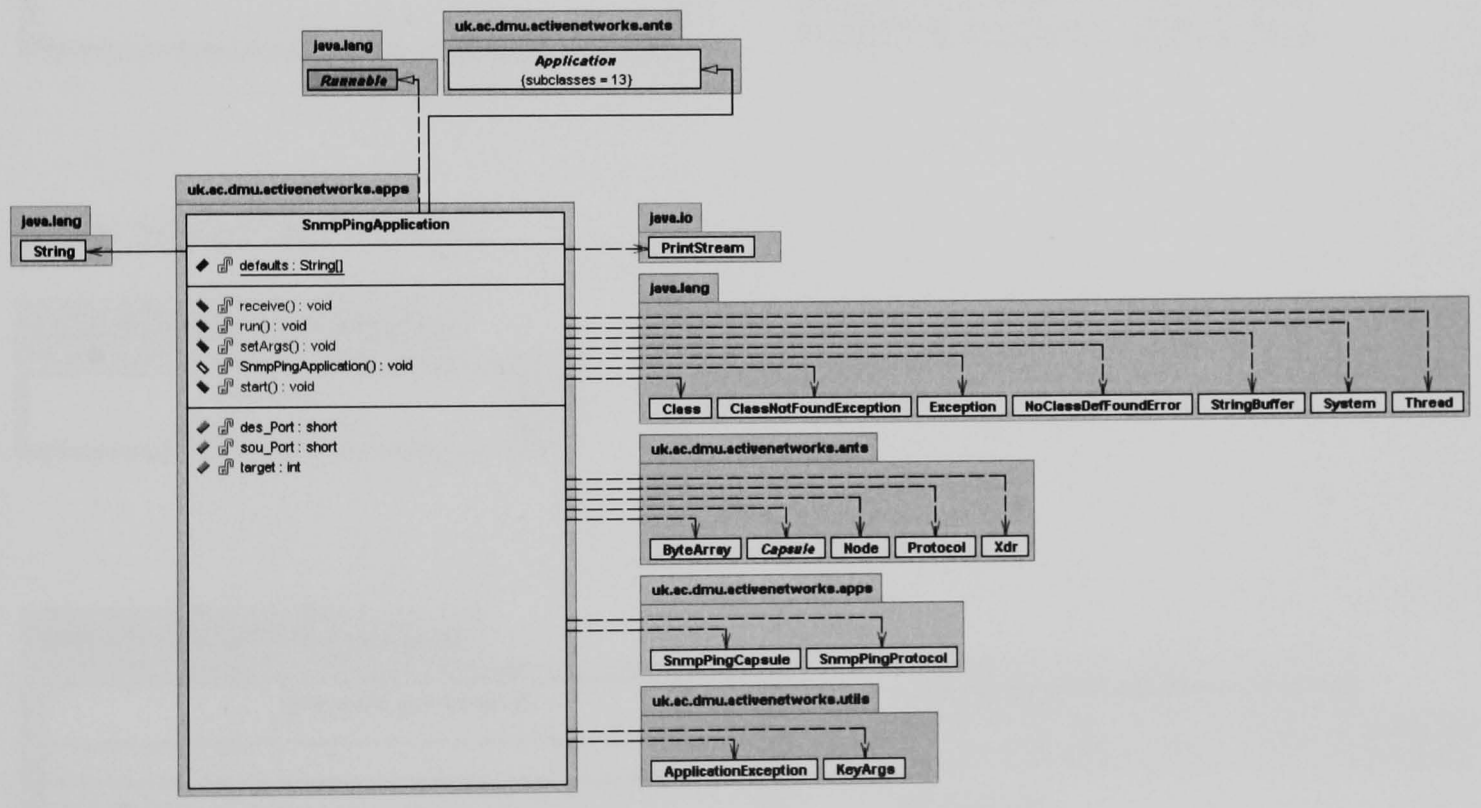
Class: SnmpMibTimeProtocol



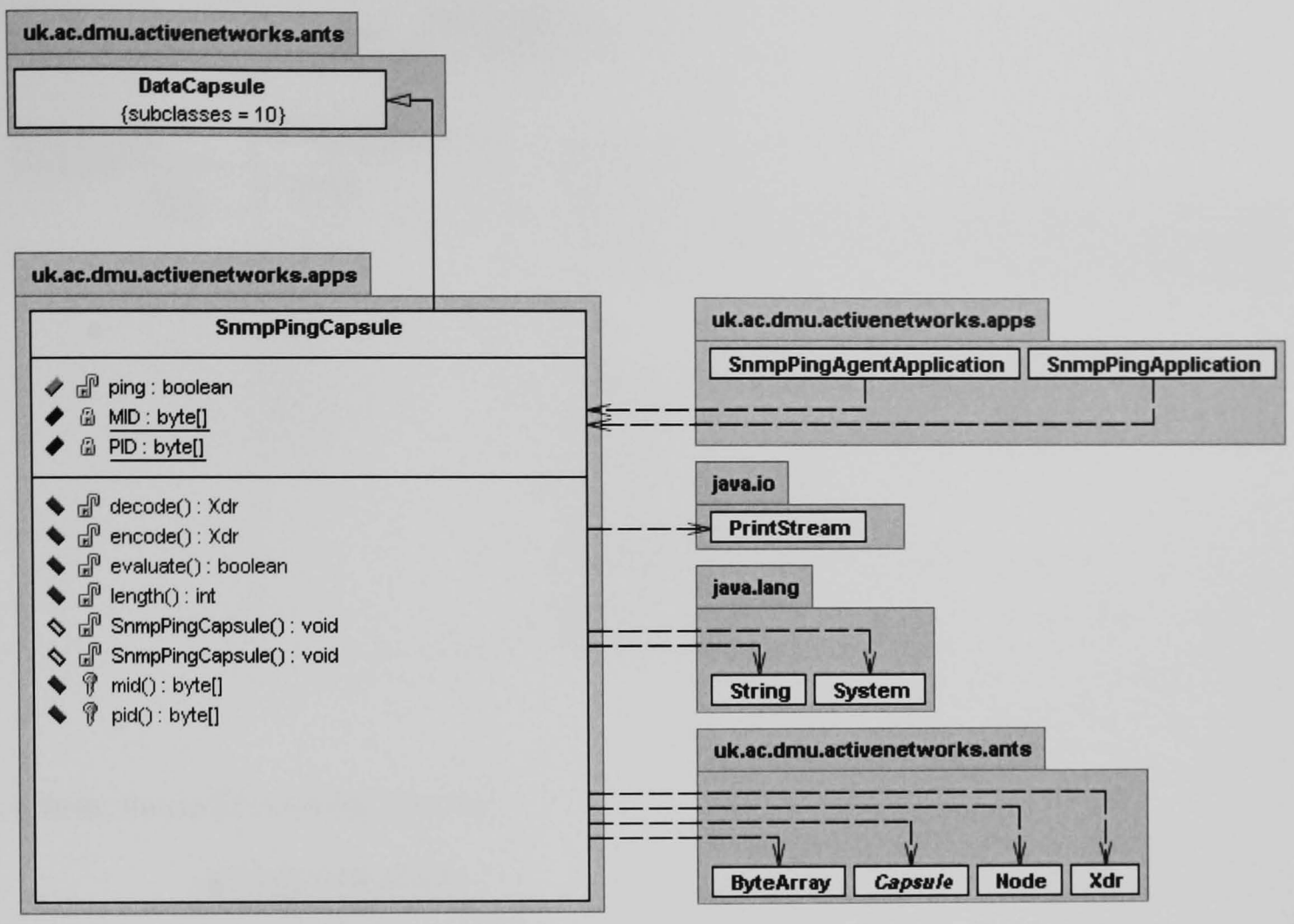
Class:SnmpPingAgentApplication



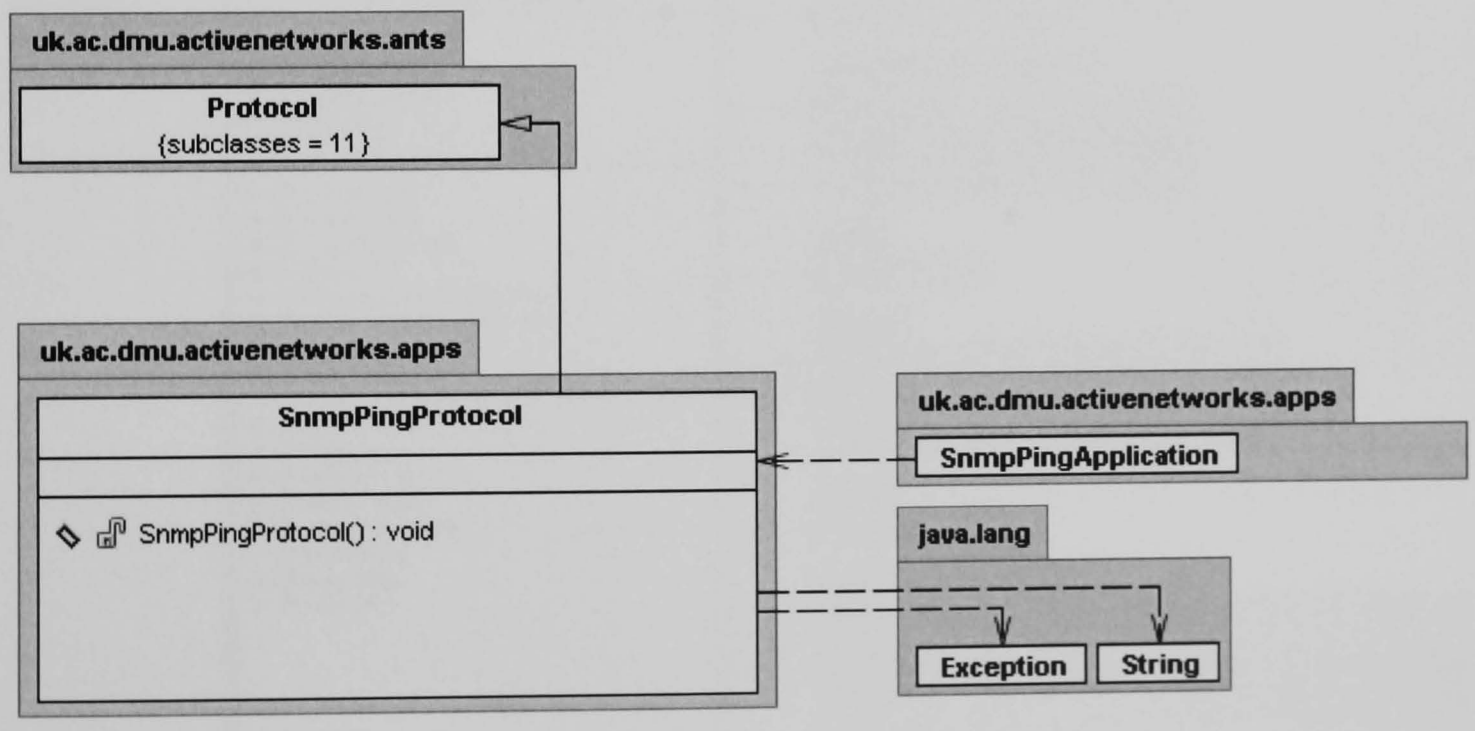
Class: SnmpPingApplication



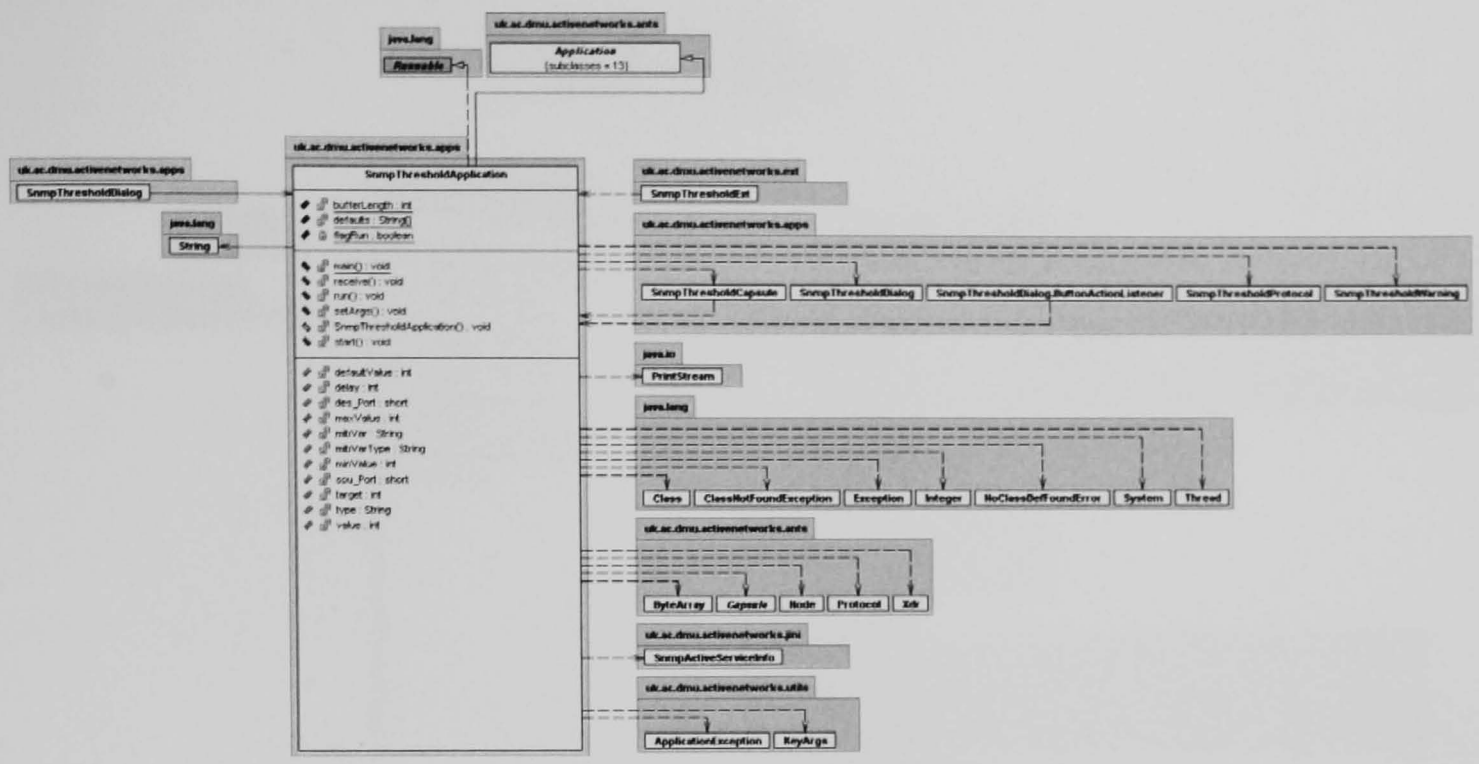
Class: SnmpPingCapsule



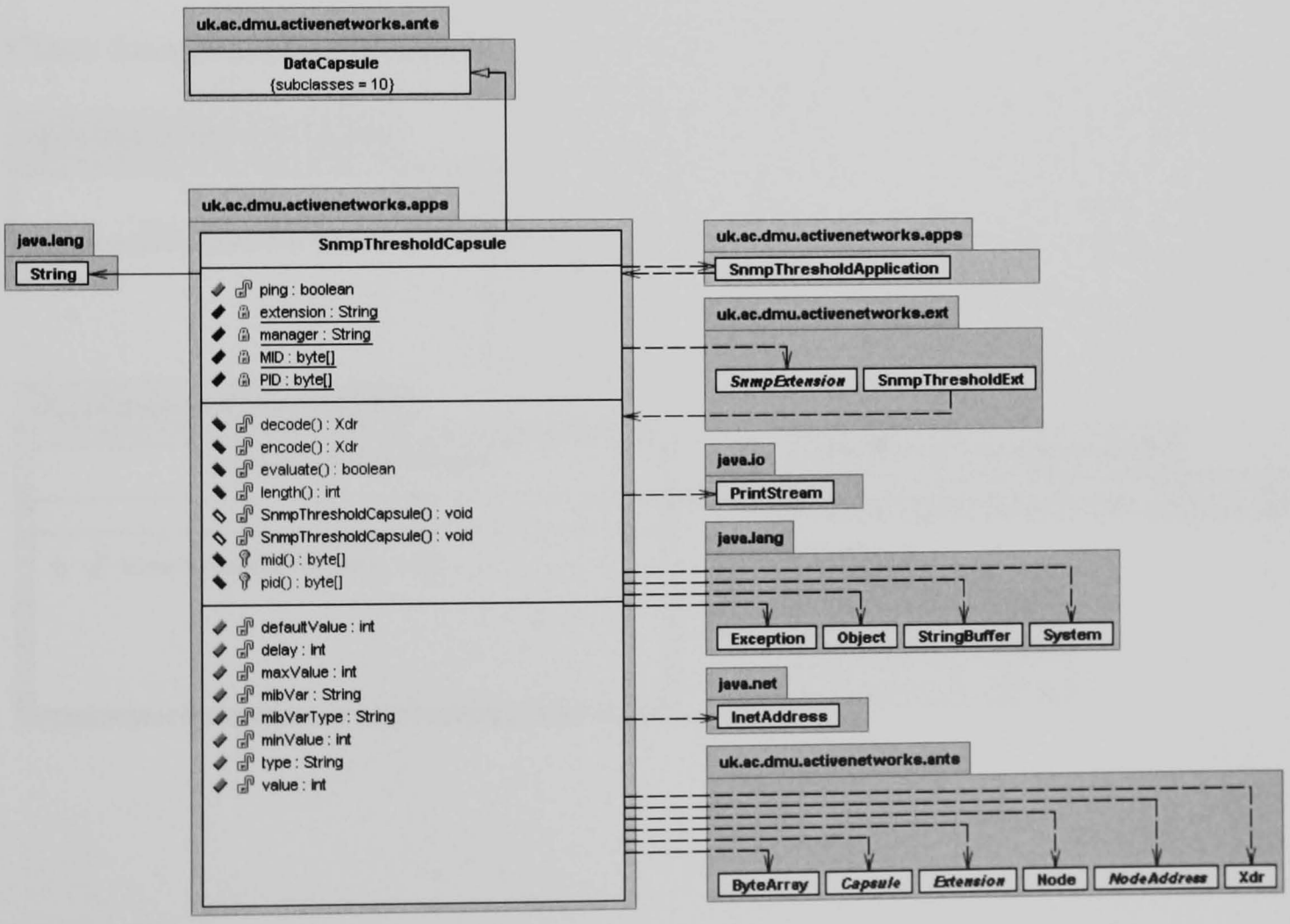
Class: SnmpPingProtocol



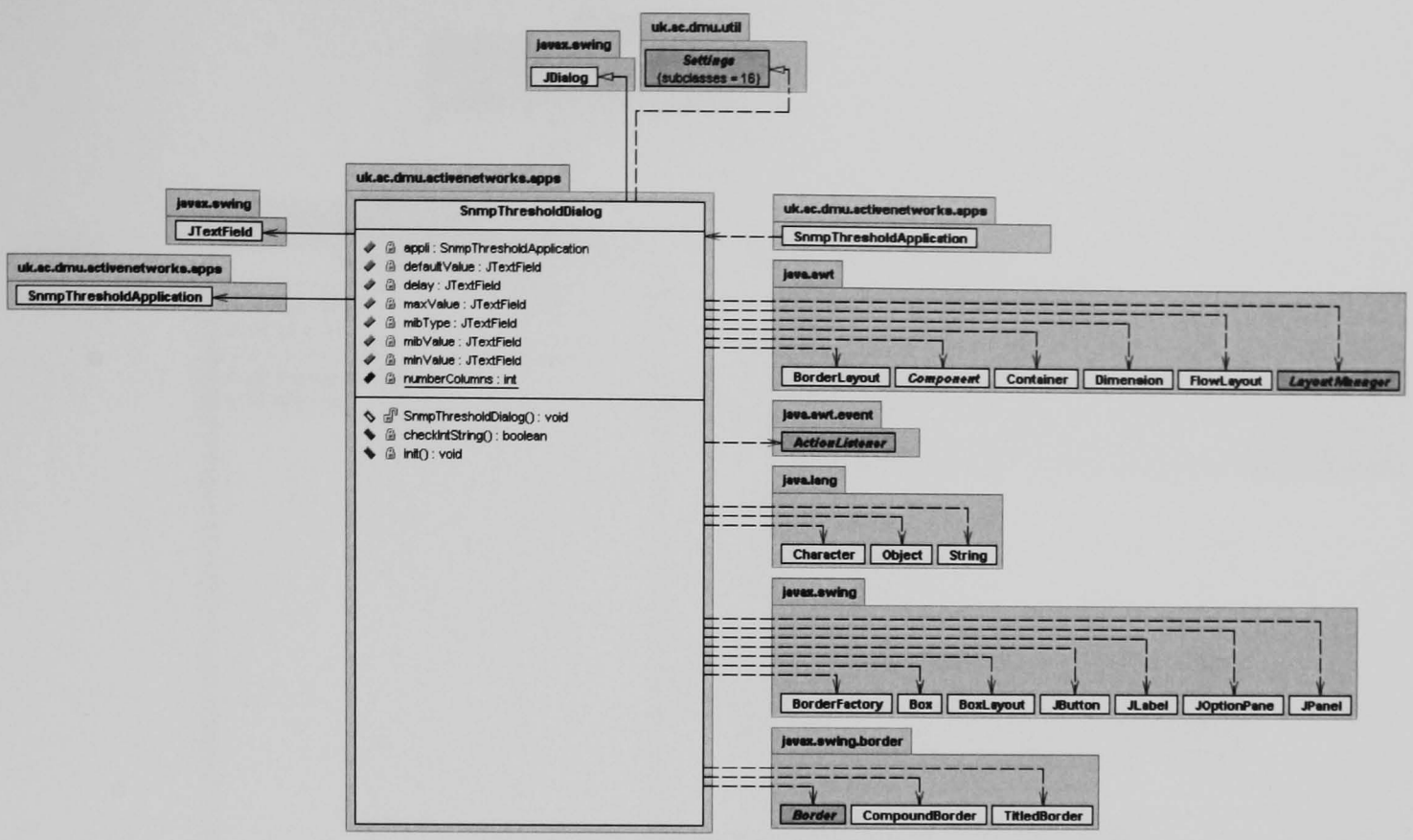
Class: SnmpThresholdApplication



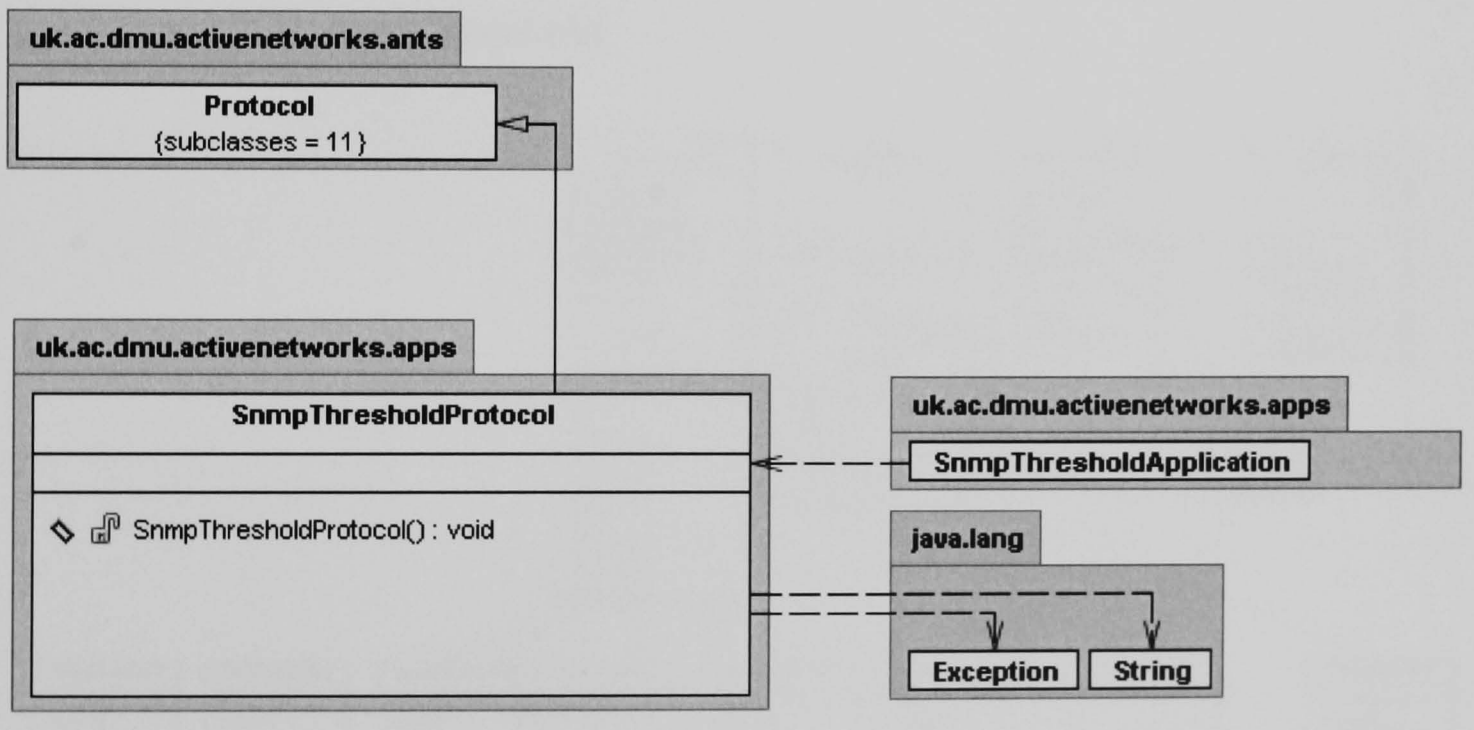
Class: SnmpThresholdCapsule



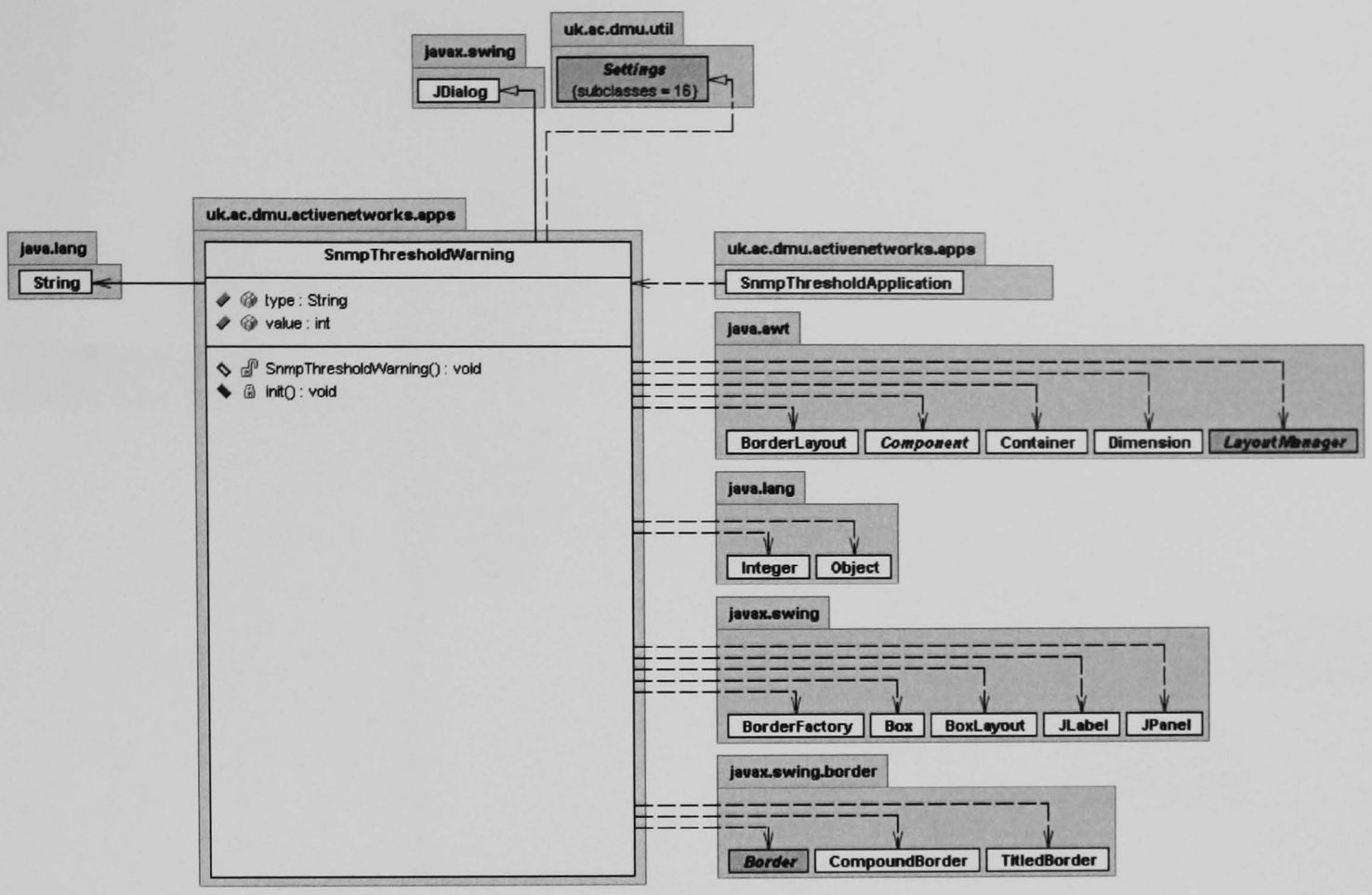
Class: SnmpThresholdDialog



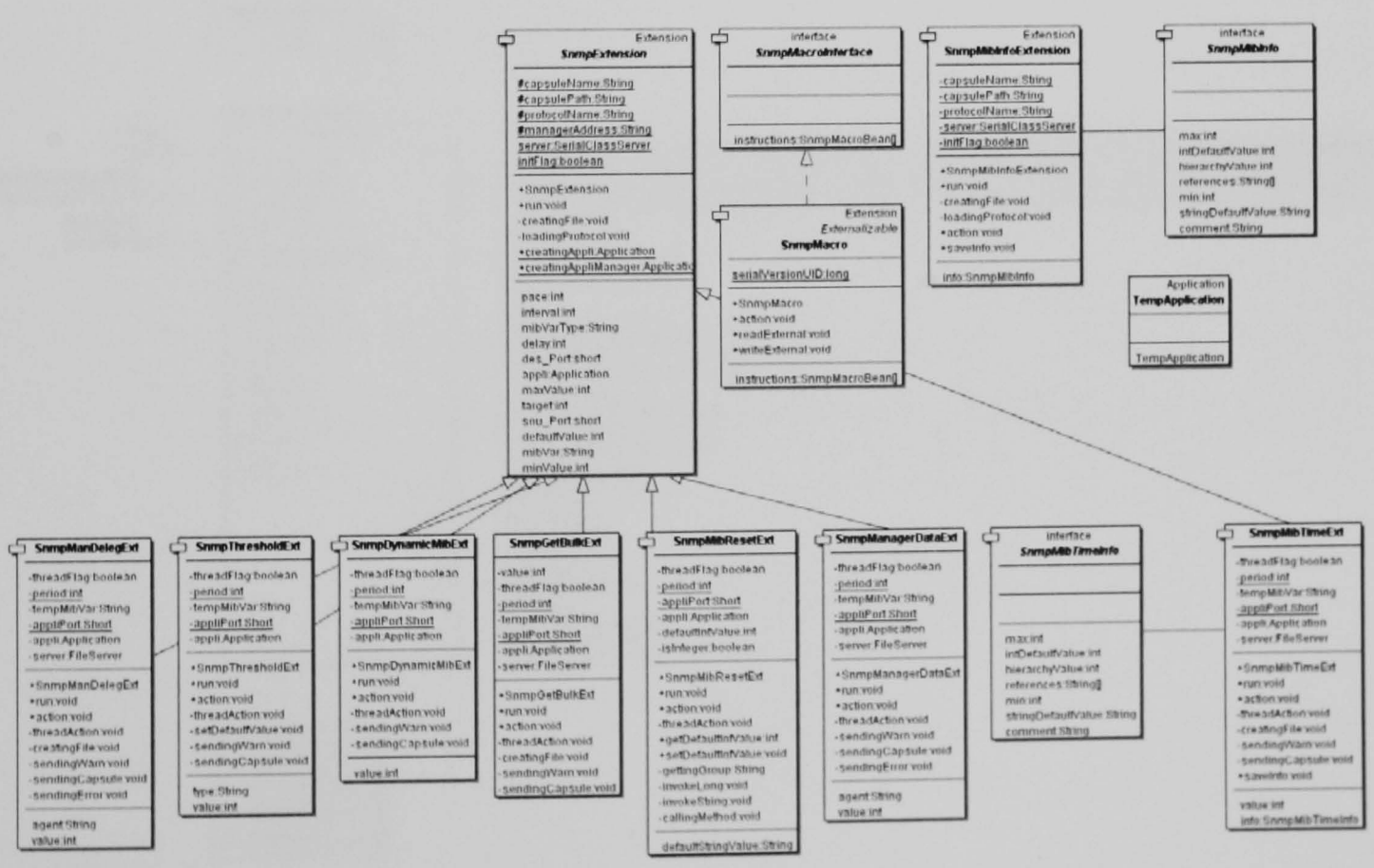
Class: SnmpThresholdProtocol



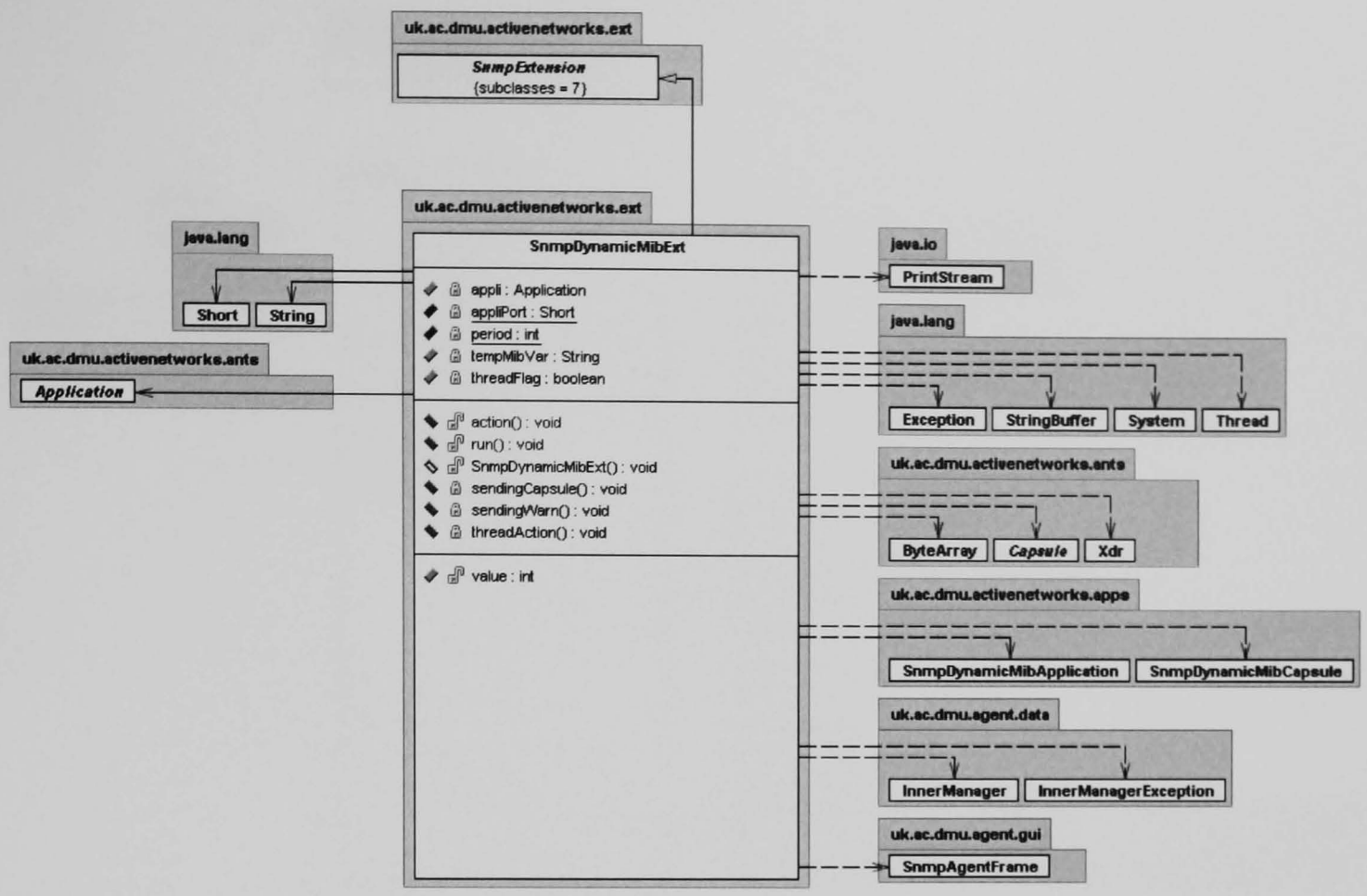
Class: SnmpThresholdWarning



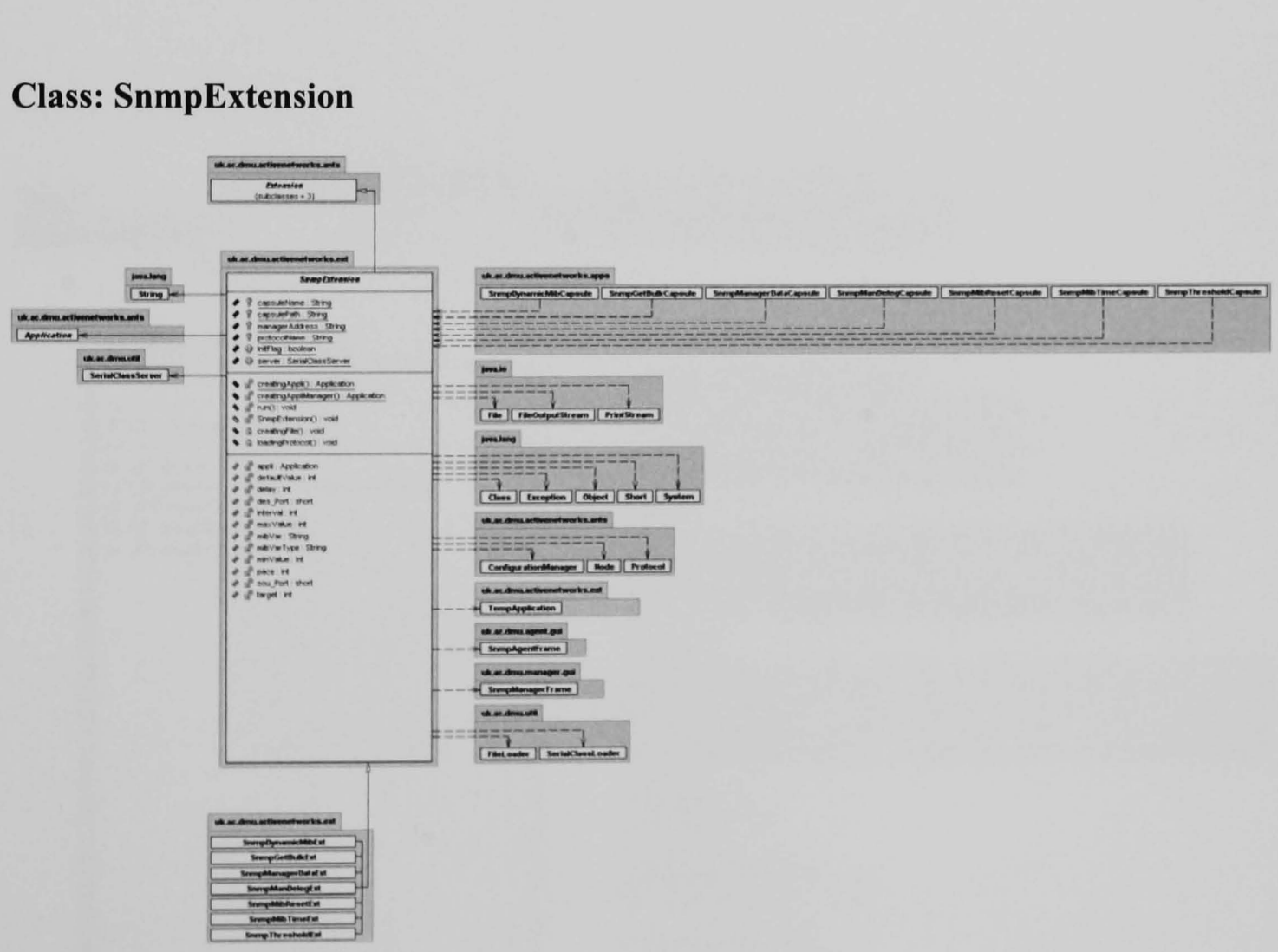
package: uk.ac.dmu.apps.ext



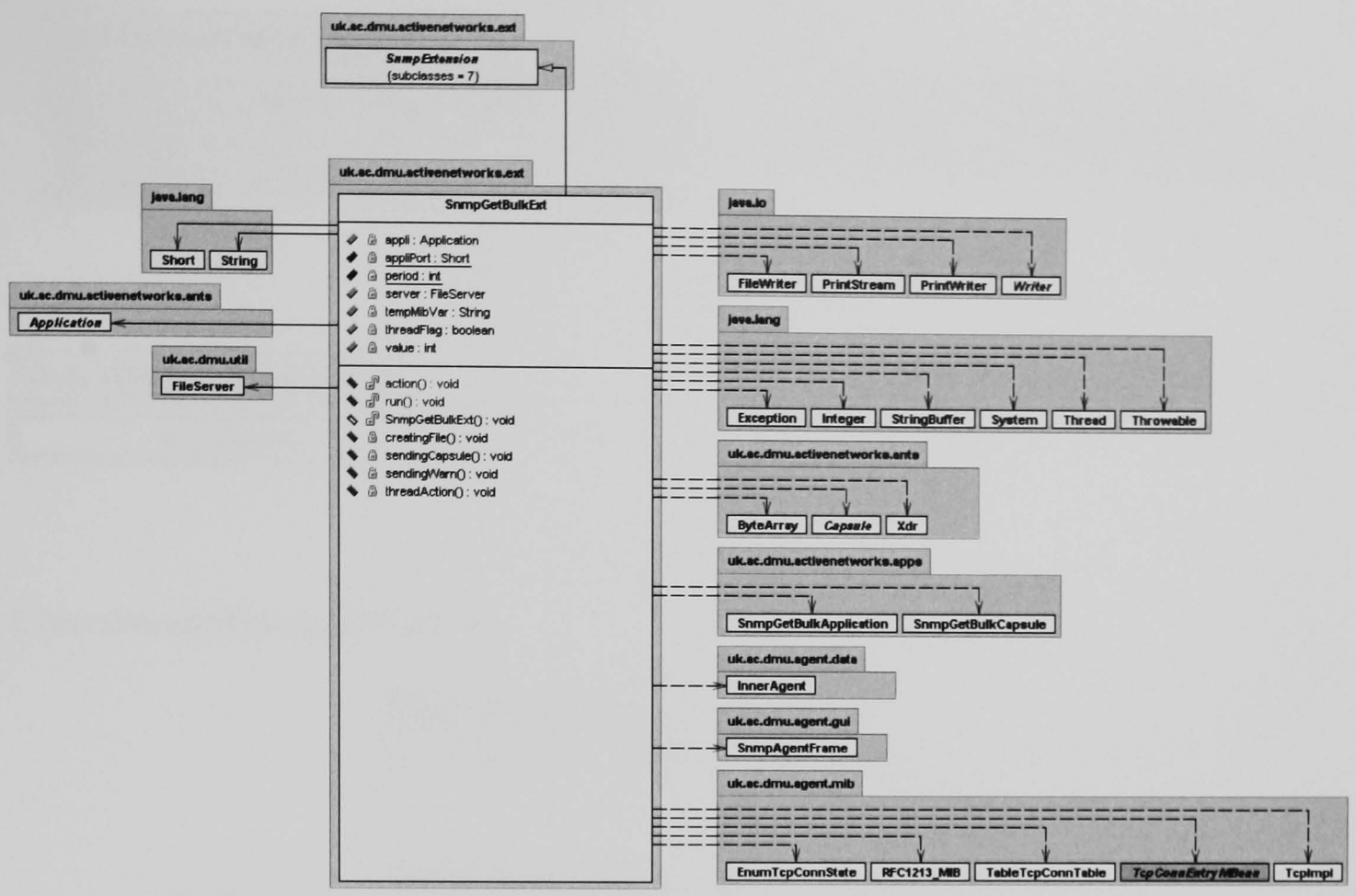
Class: SnmpDynamicMibExt



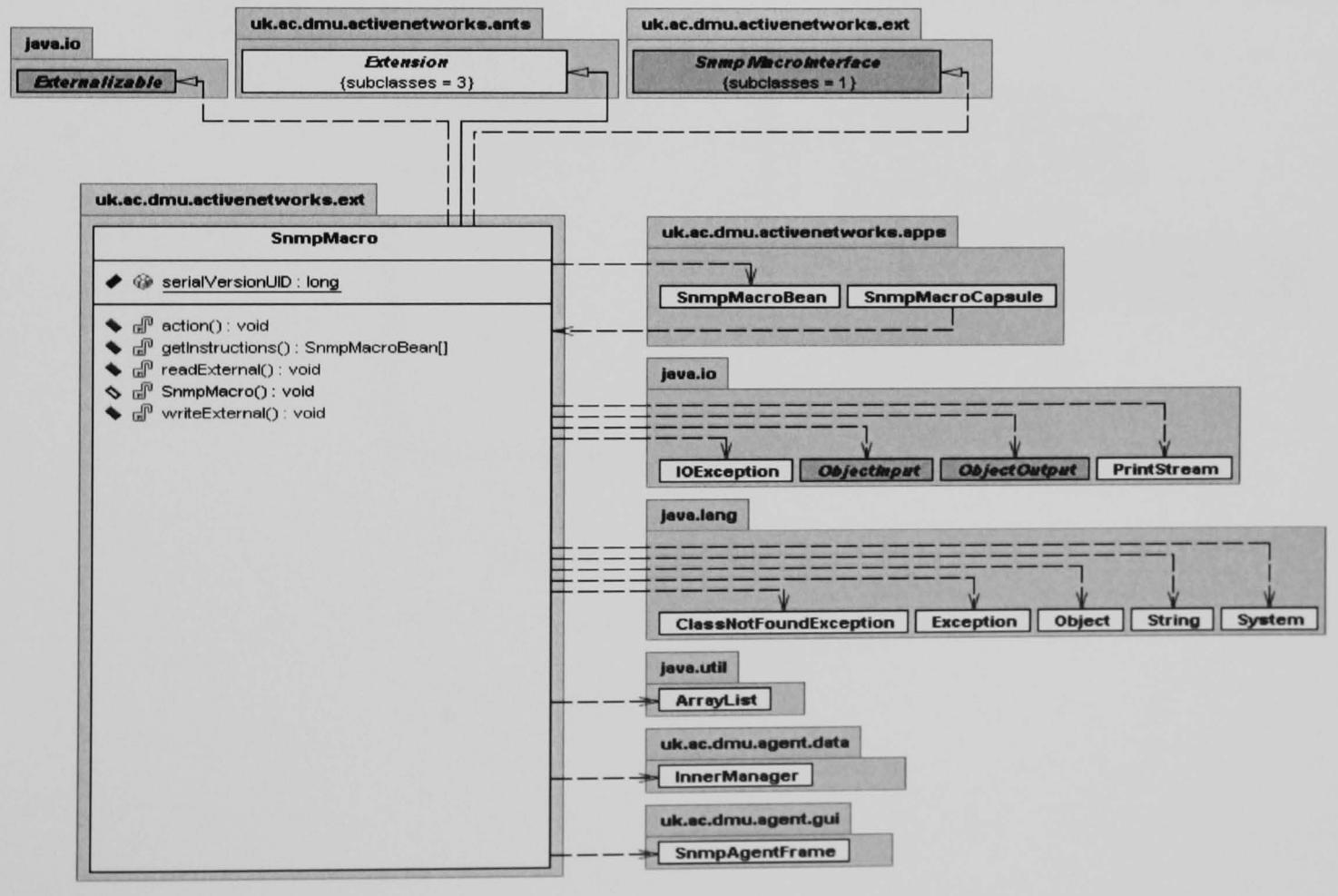
Class: SnmpExtension



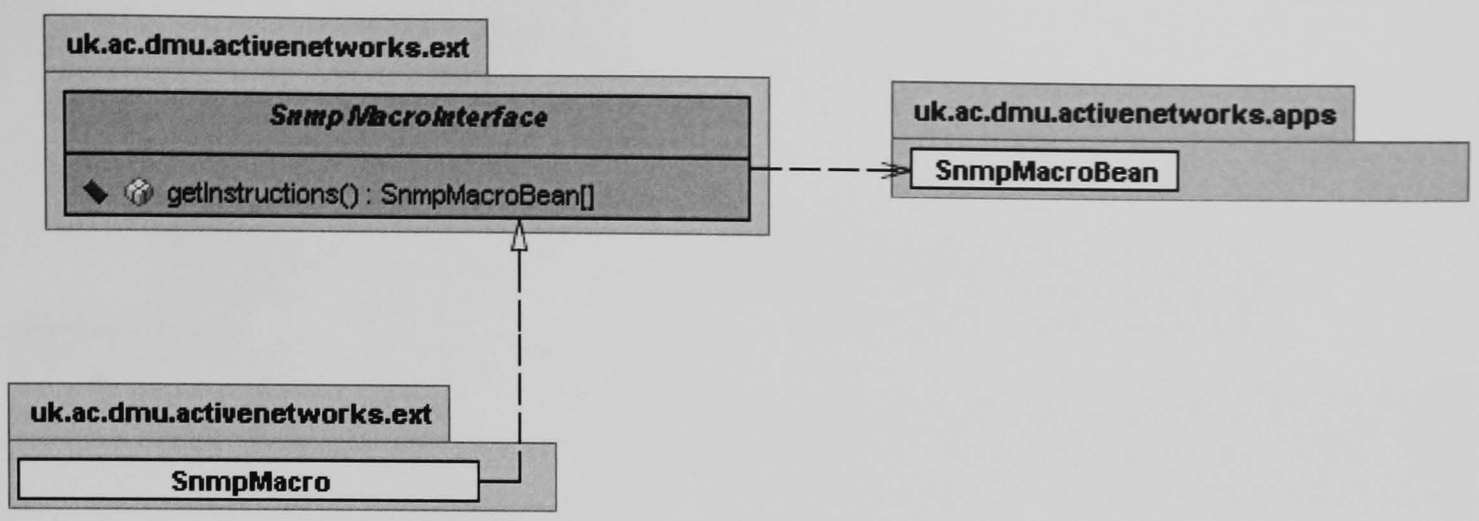
Class: SnmpGetBulkExt



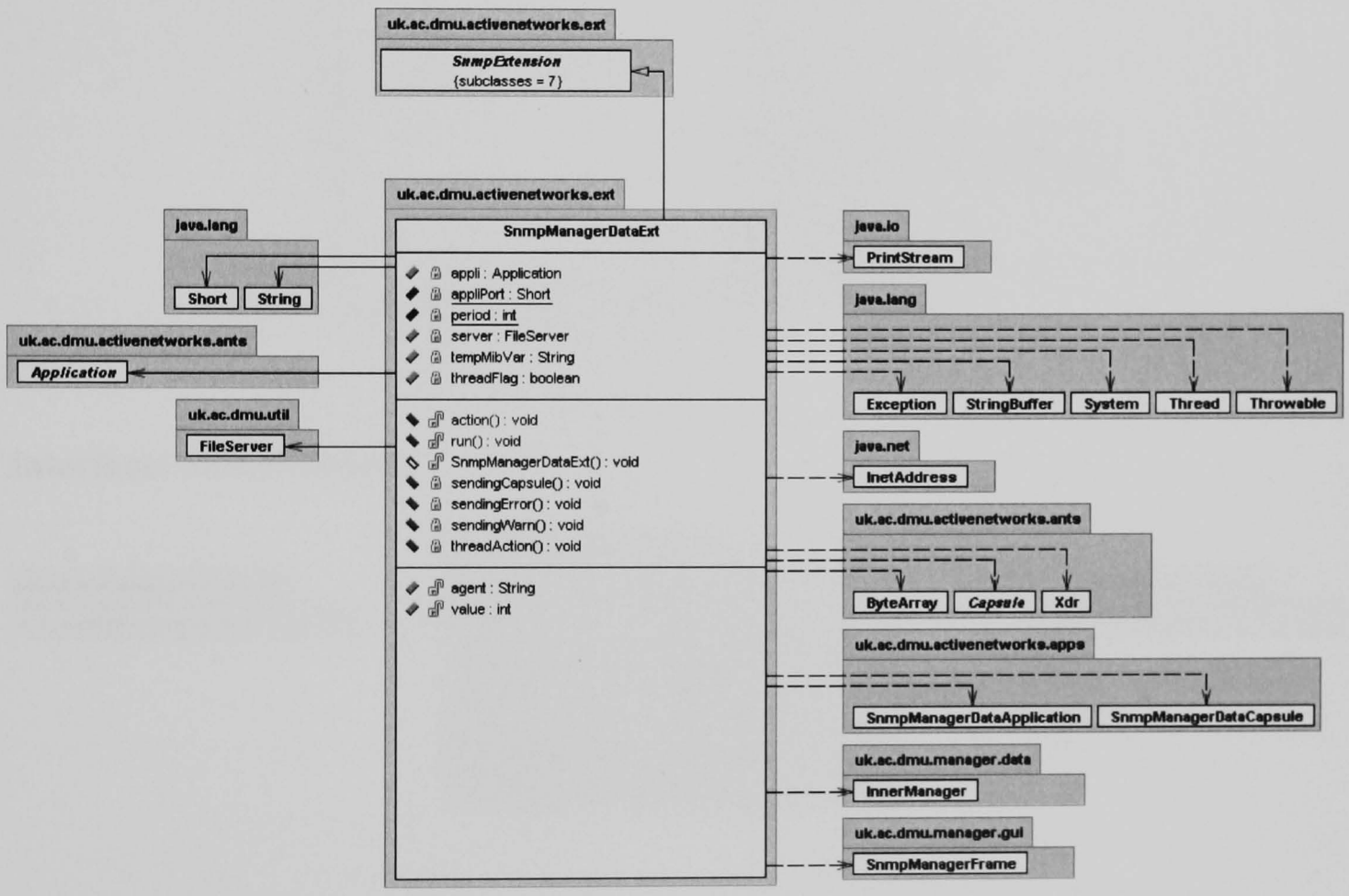
Class: SnmpMacro



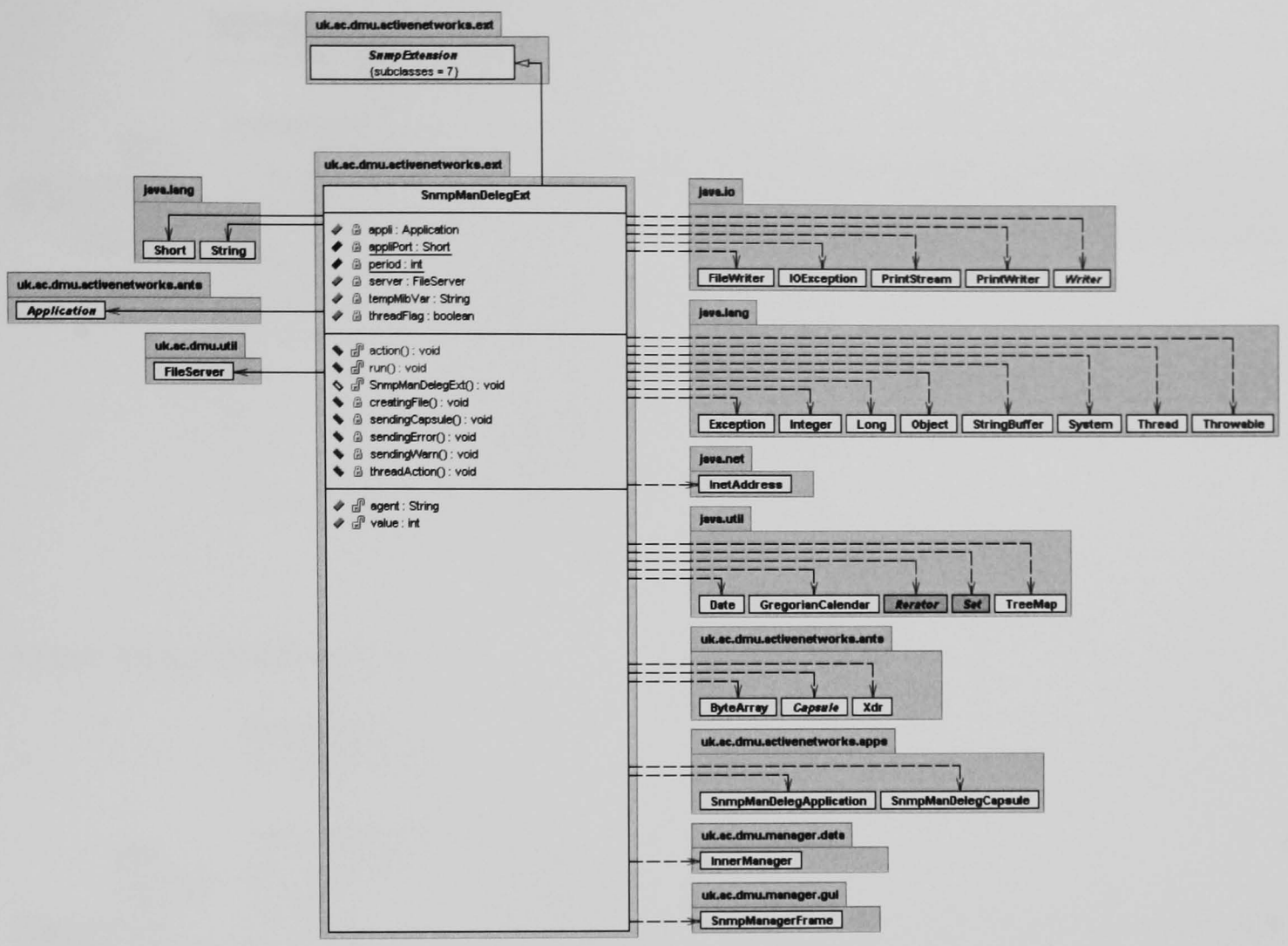
Interface: SnmpMacroInterface



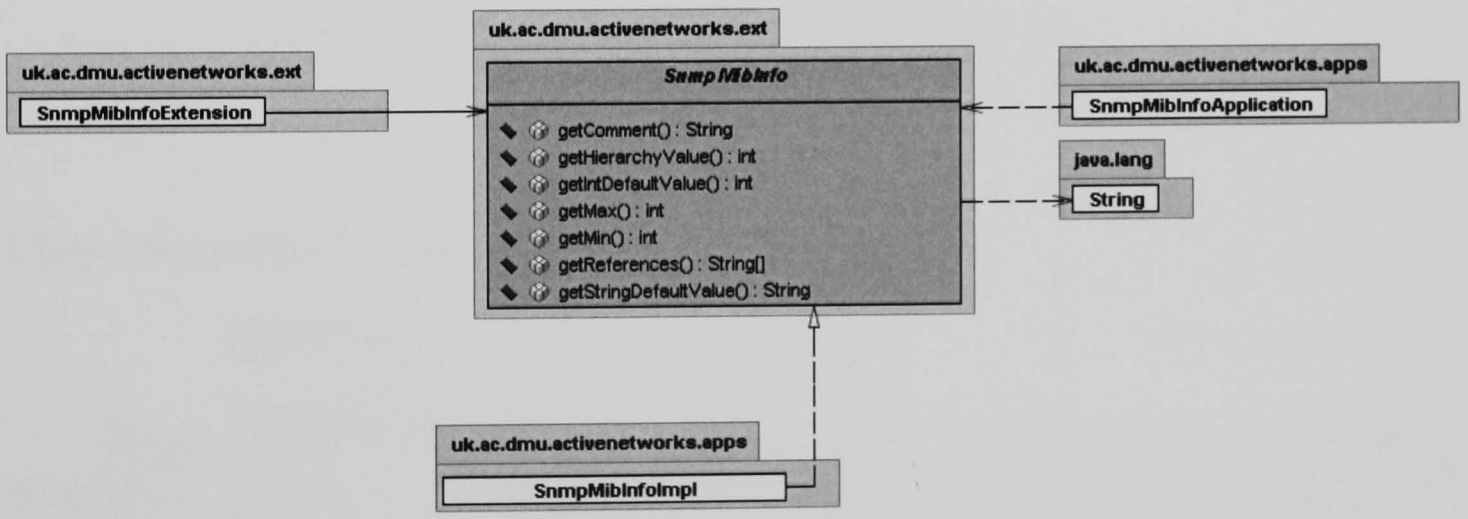
Class:SnmpManagerDataExt



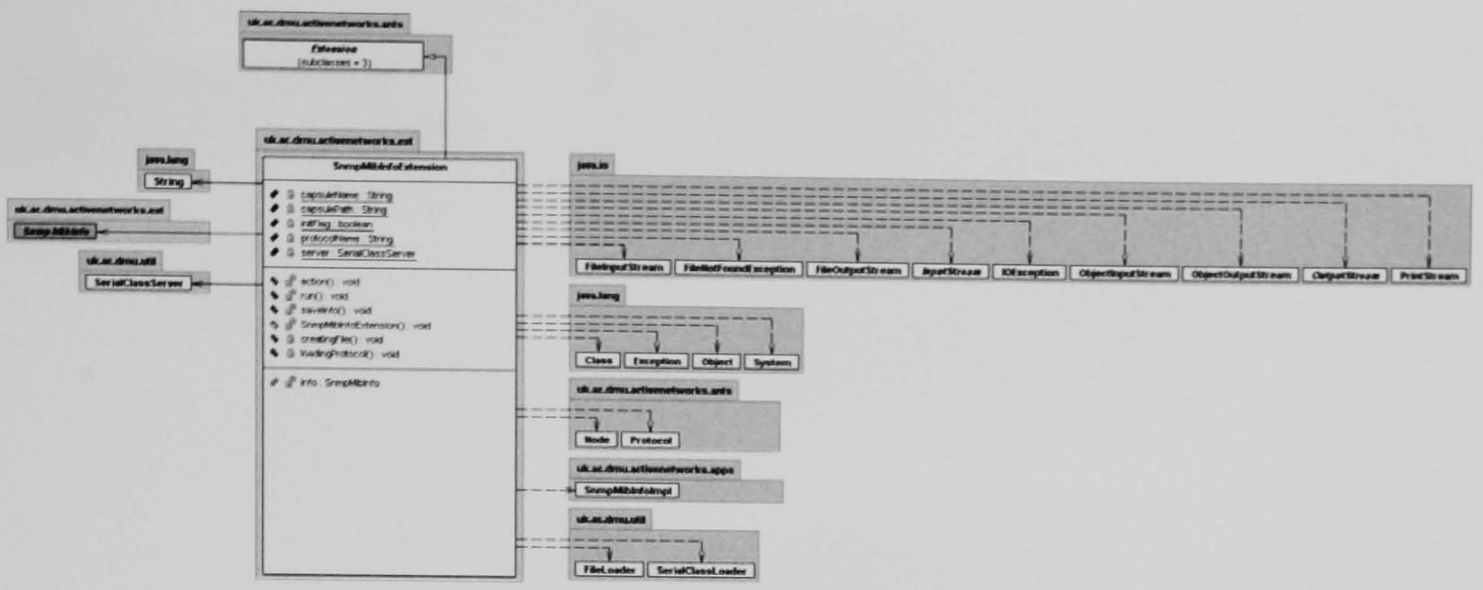
Class: SnmpManDelegExt



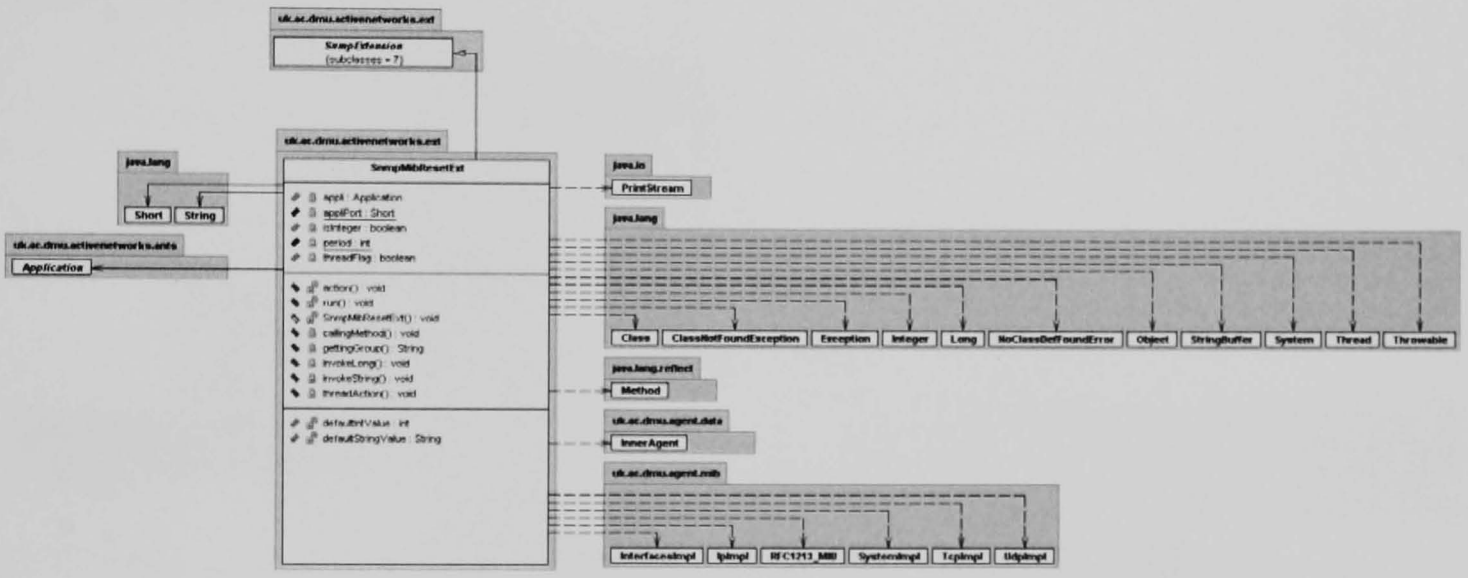
Interface: SnmpMibInfo



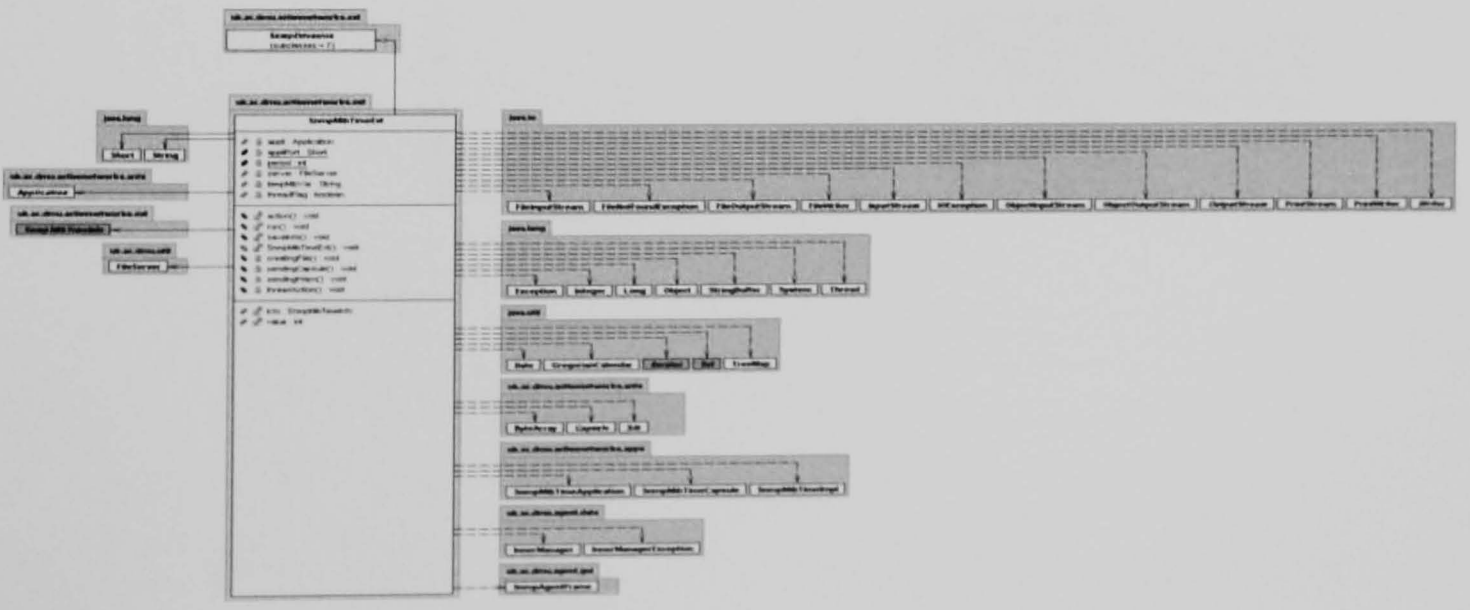
Class:SnmpMibInfoExtension



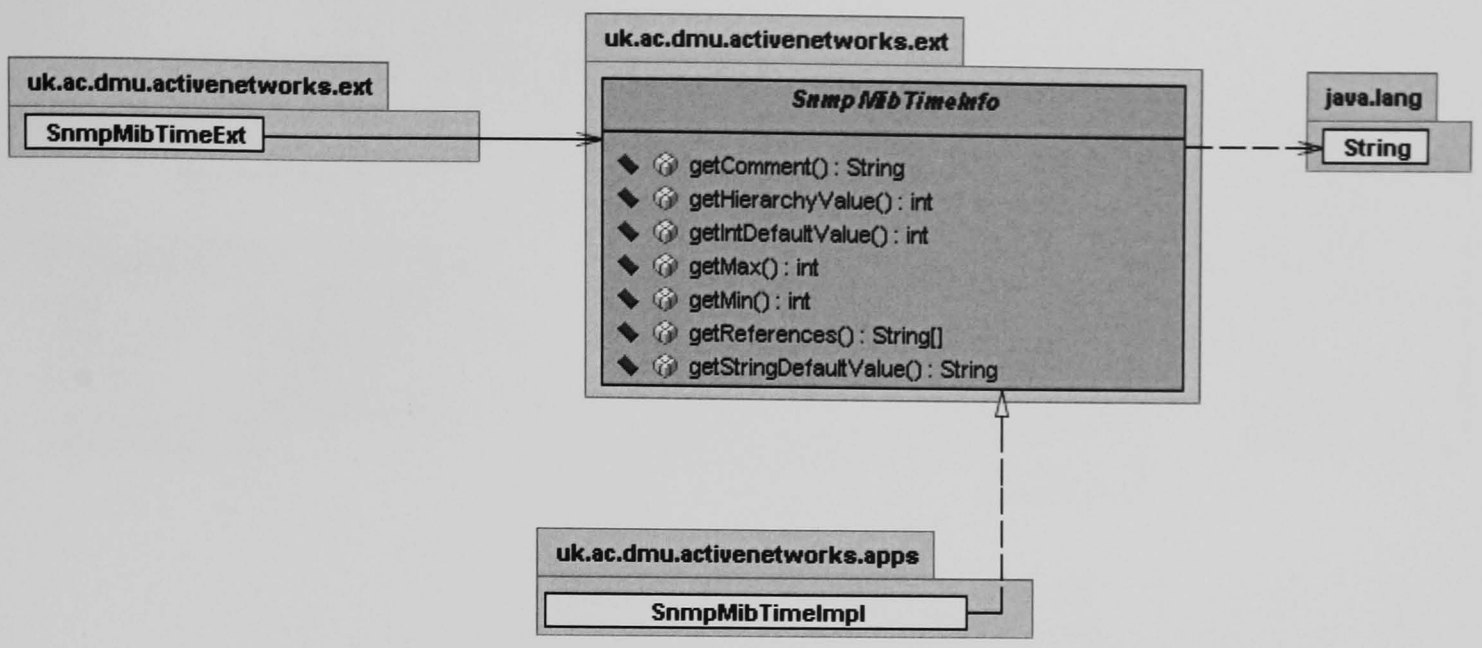
Class: SnmpMibResetExt



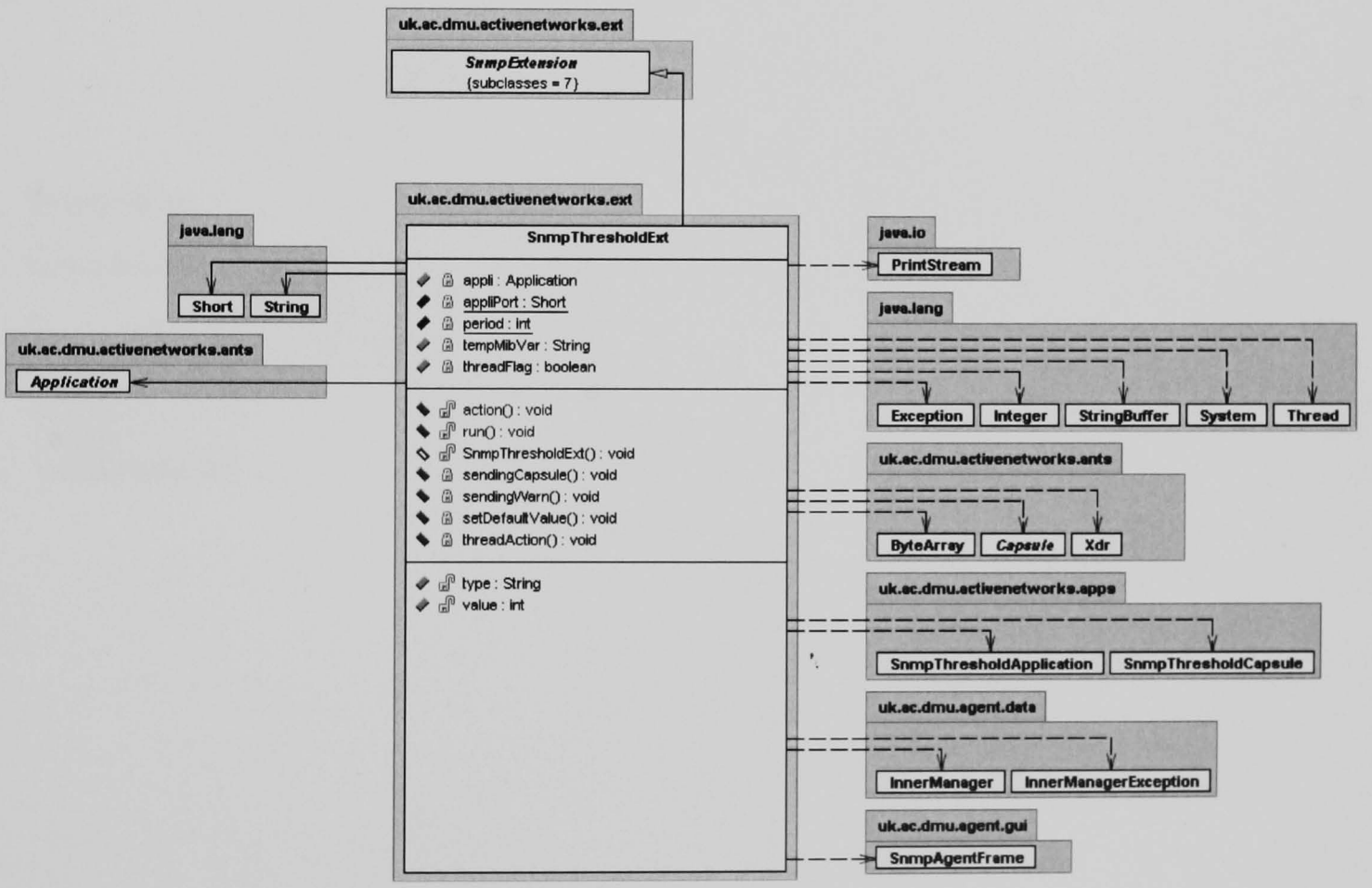
Class: SnmpMibTimeExt



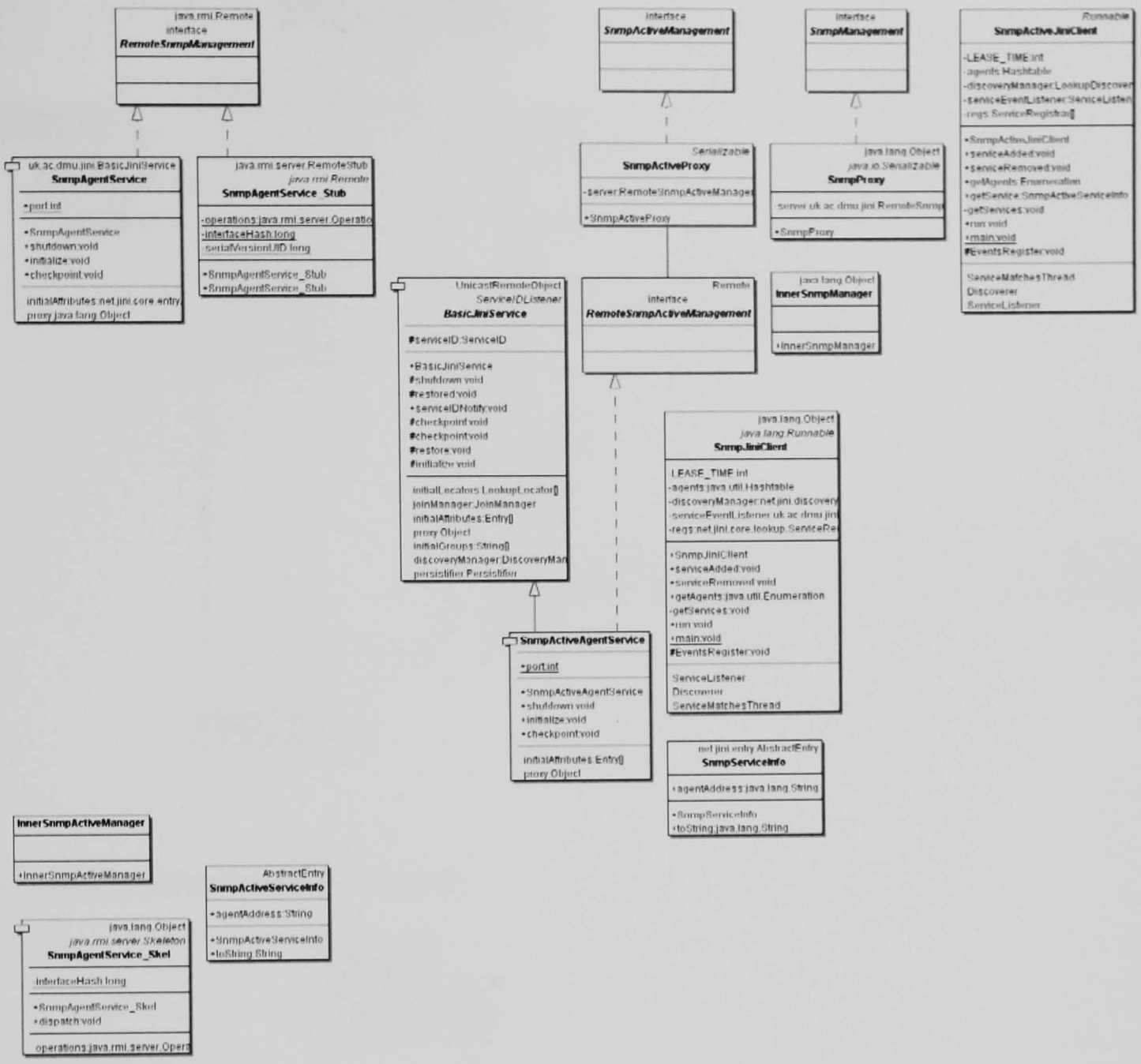
Interface: SnmpMibTimeInfo



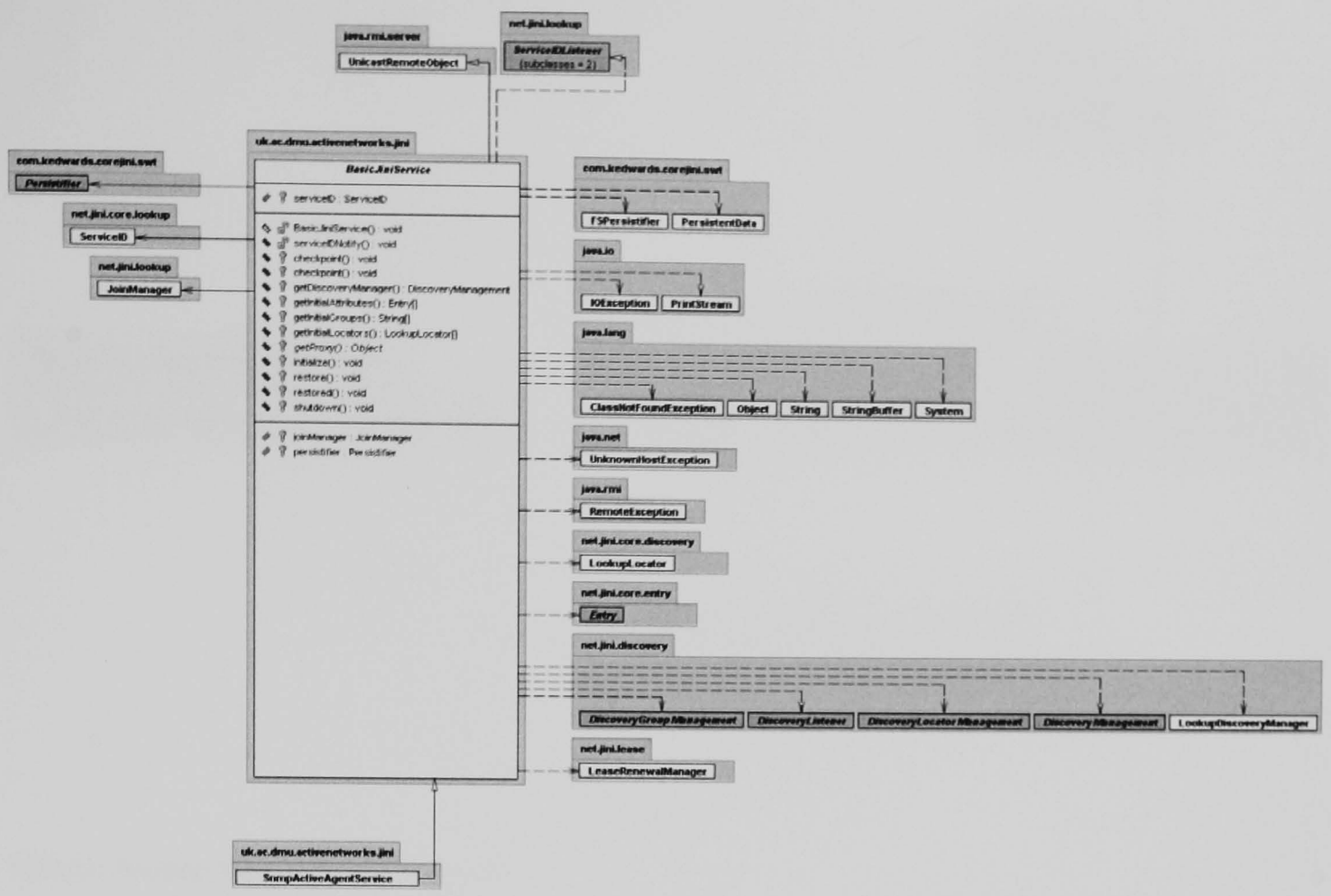
Class: SnmpThresholdExt



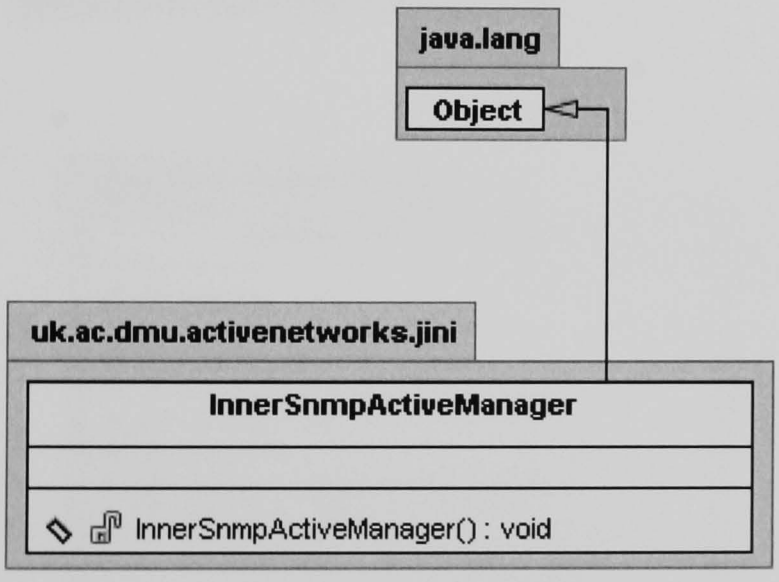
package uk.ac.dmu.activenetworks.jini



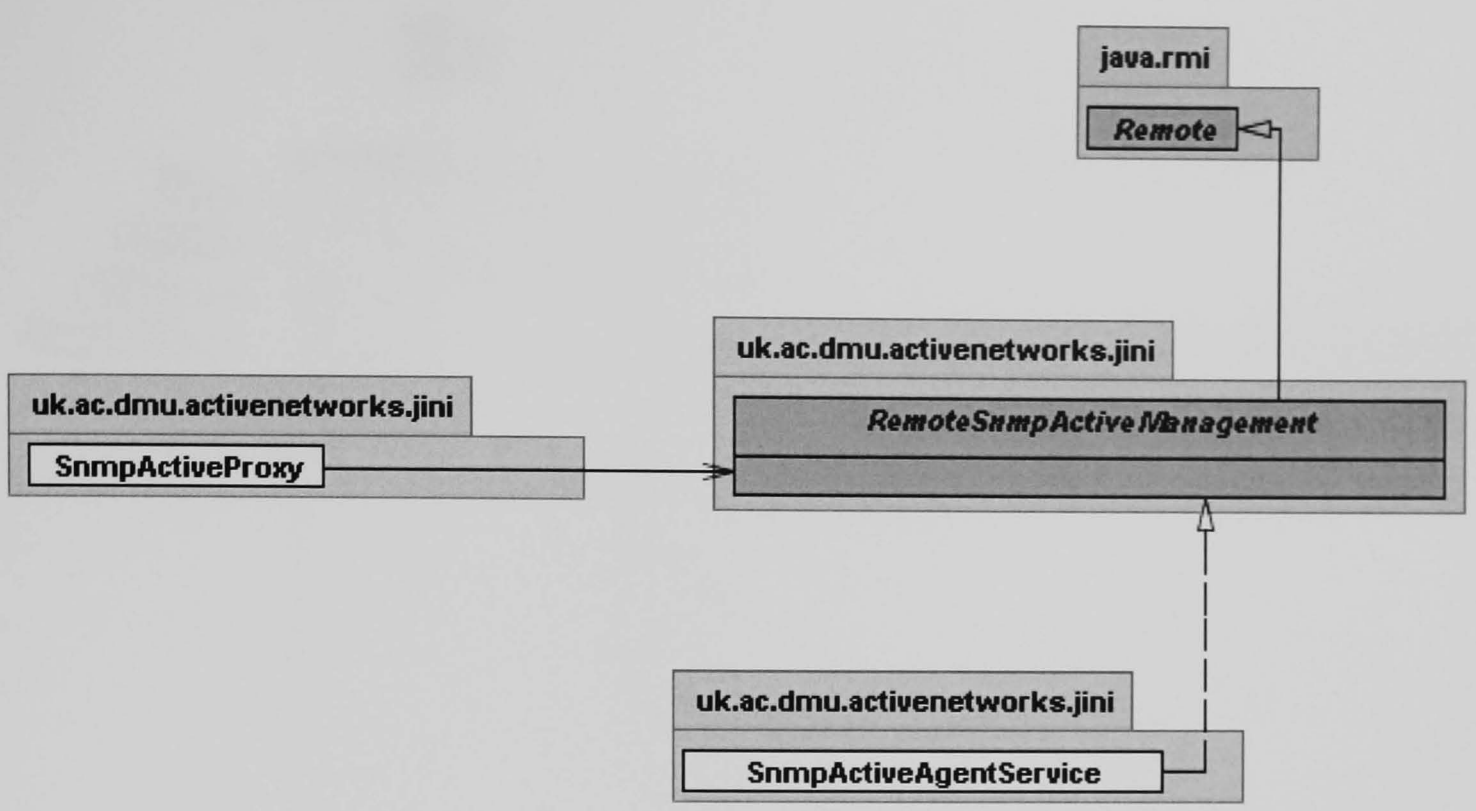
Class: BasicJiniService



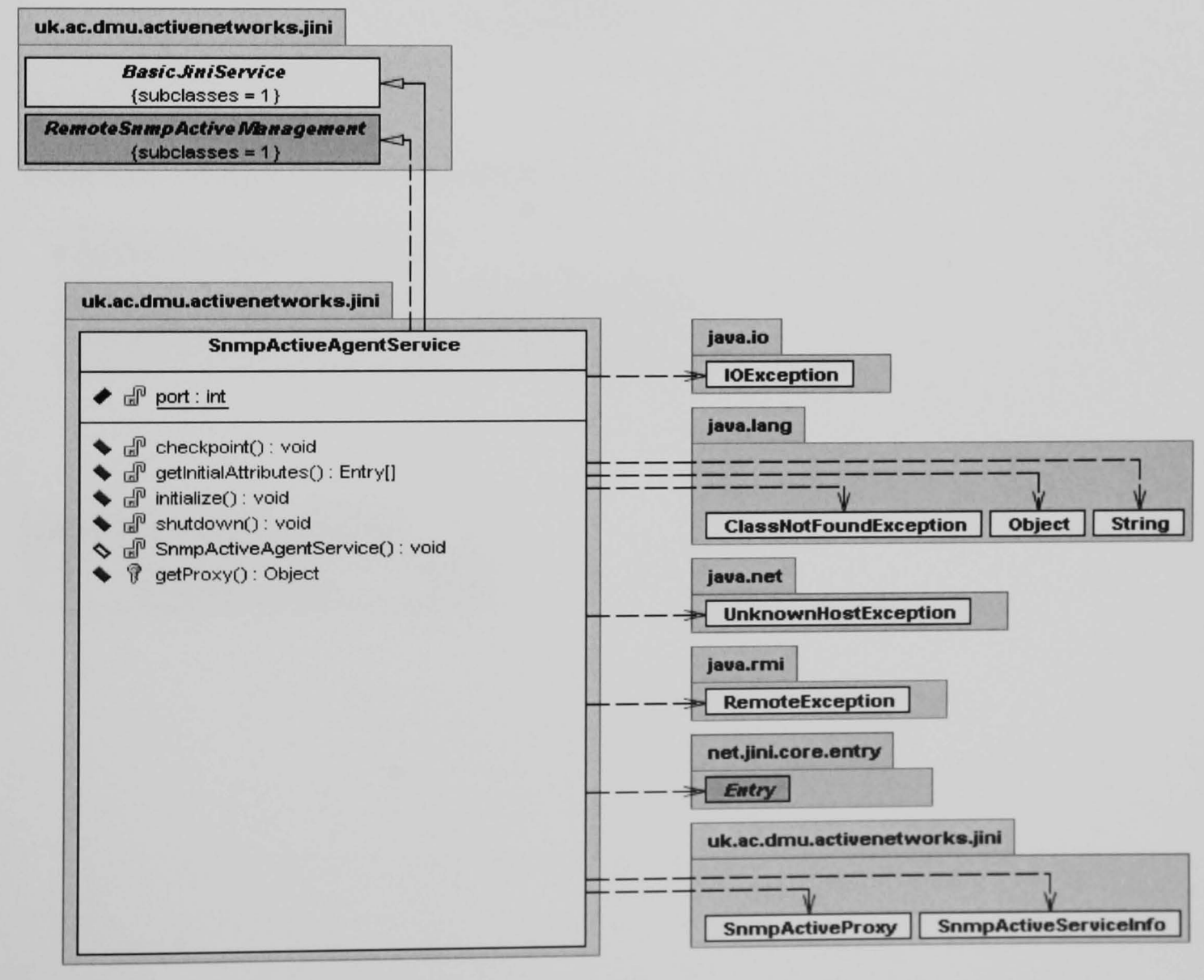
Class: InnerSnmpActiveManager



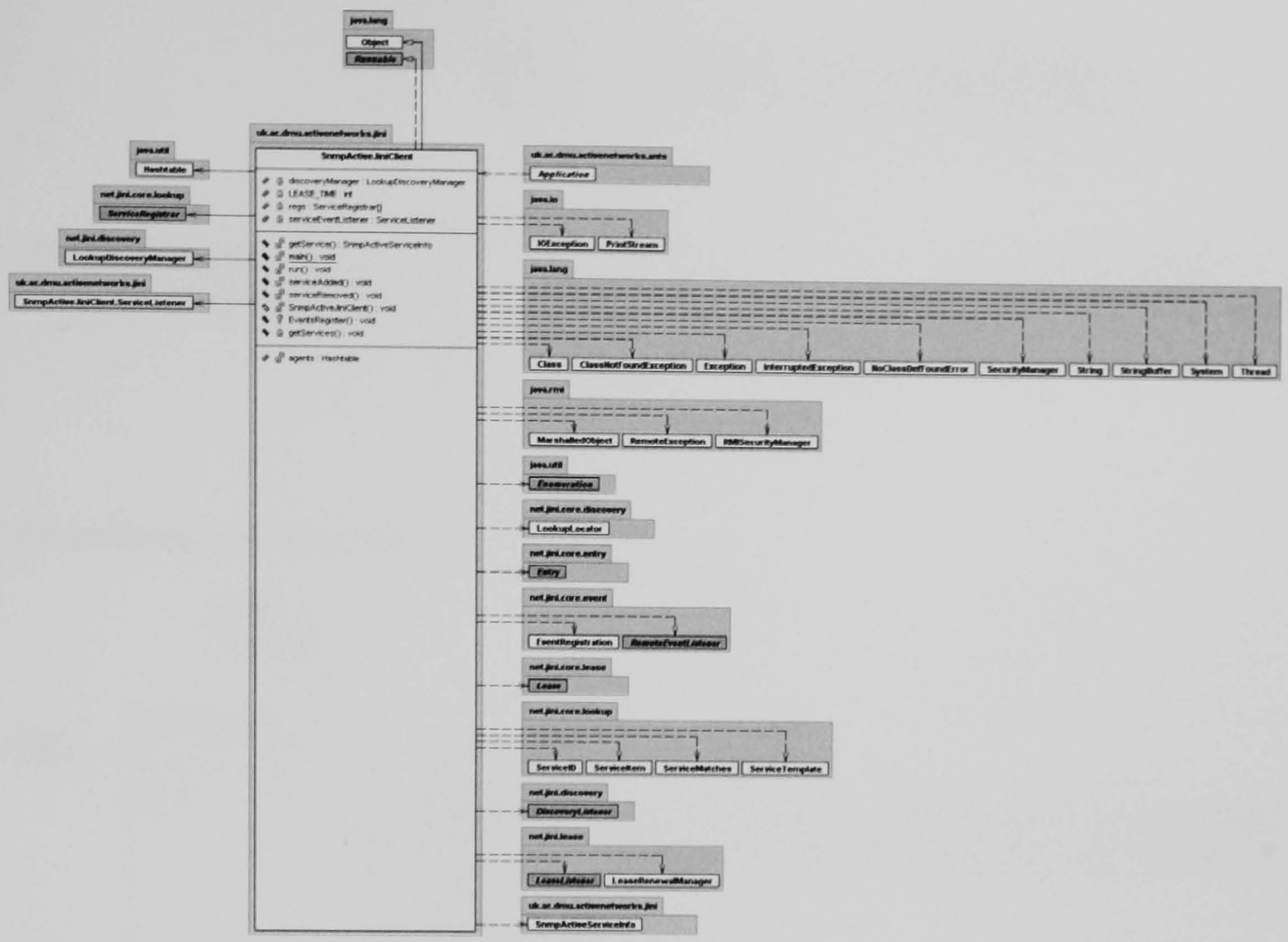
Class: RemoteSnmpActiveManagement



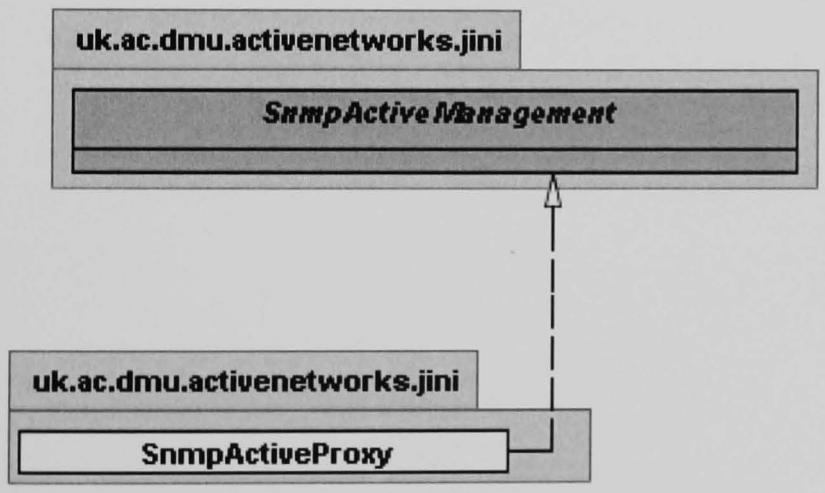
Class: SnmpActiveAgentService



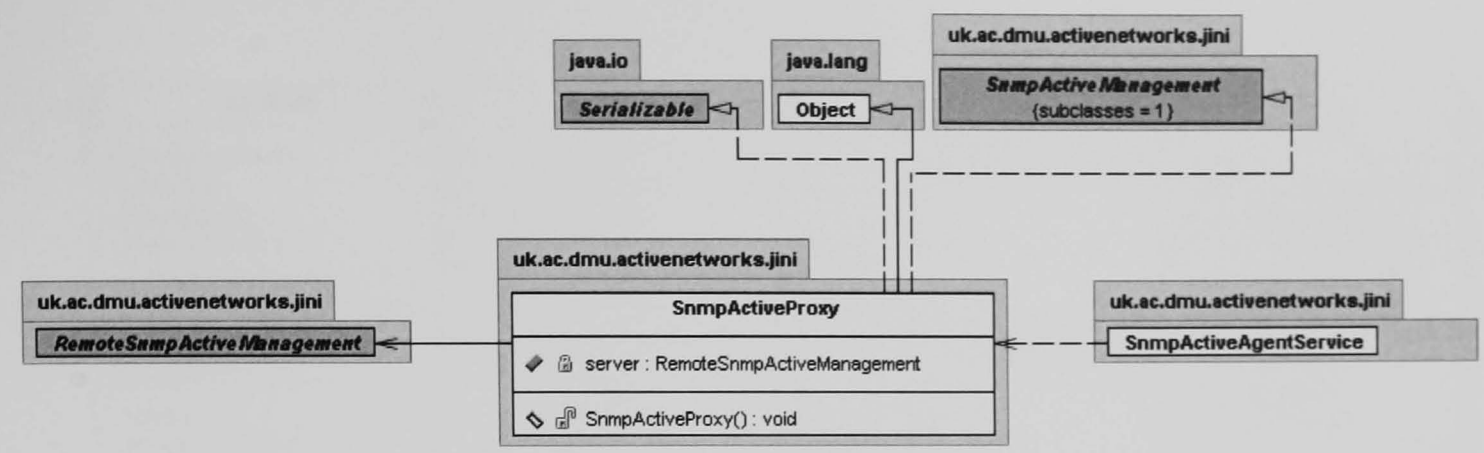
Class: SnmpActiveJiniClient



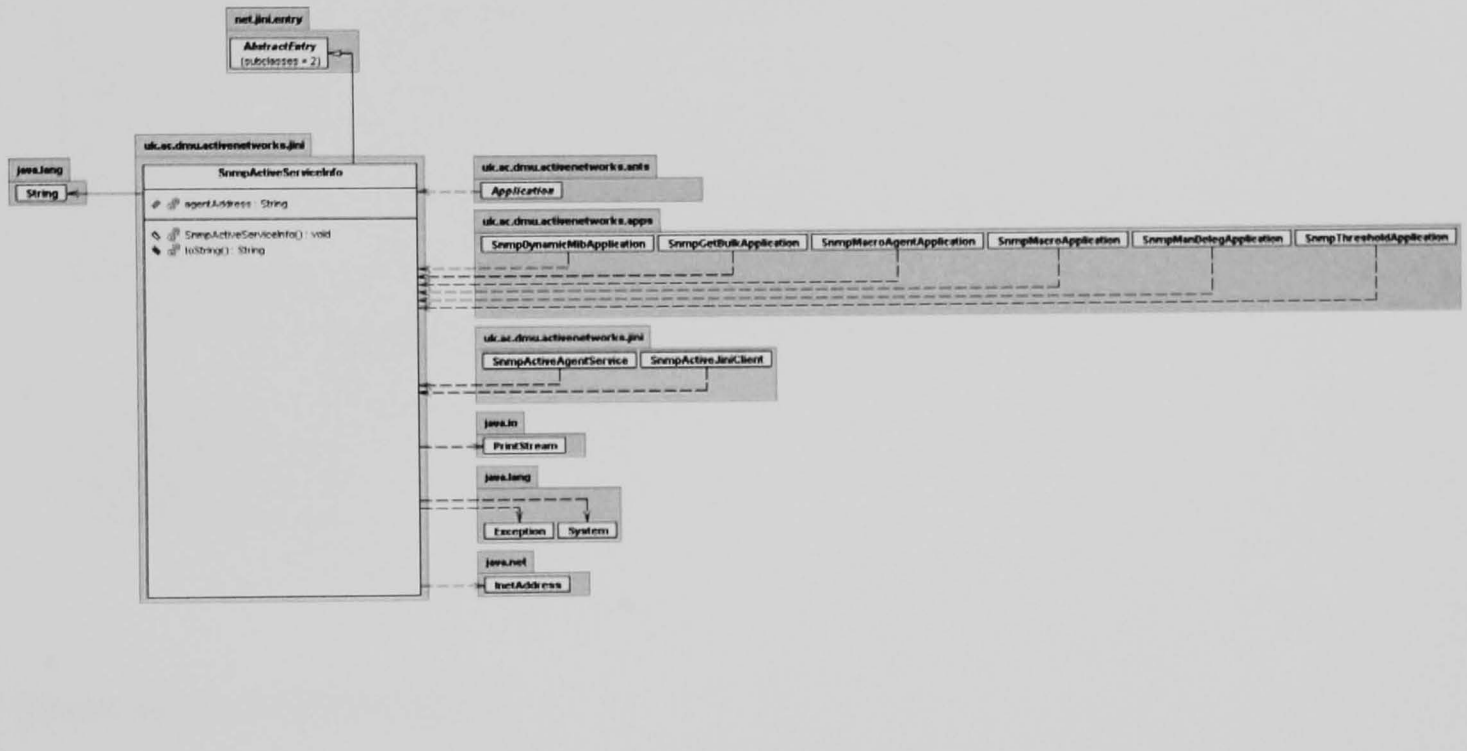
Interface: SnmpActiveManagement



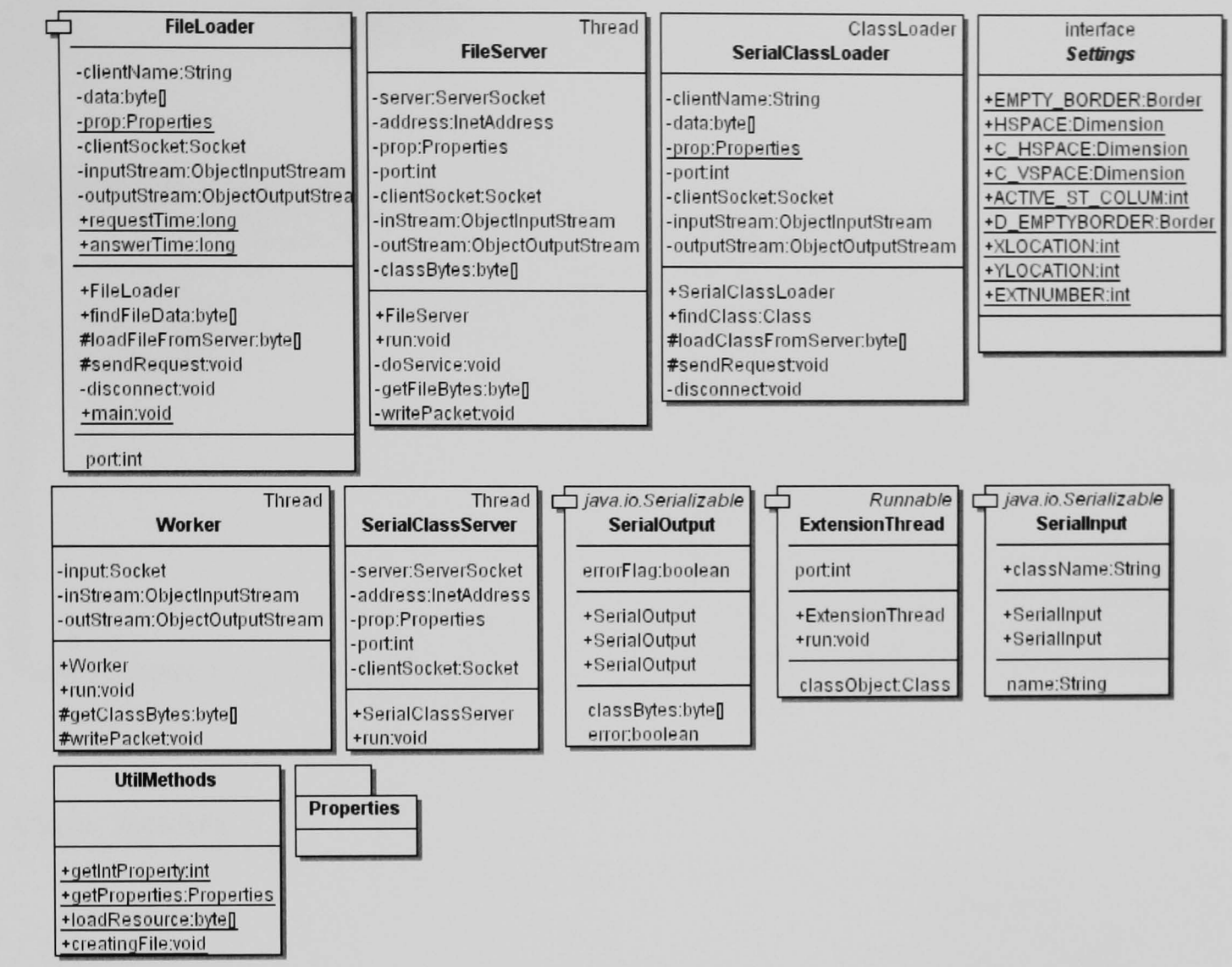
Class:SnmpActiveProxy



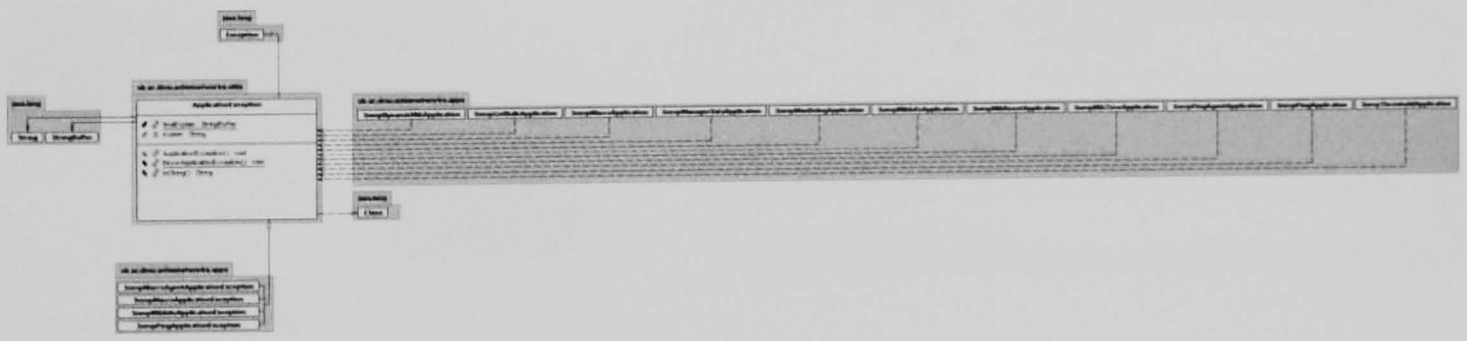
Class:SnmpActiveServiceInfo



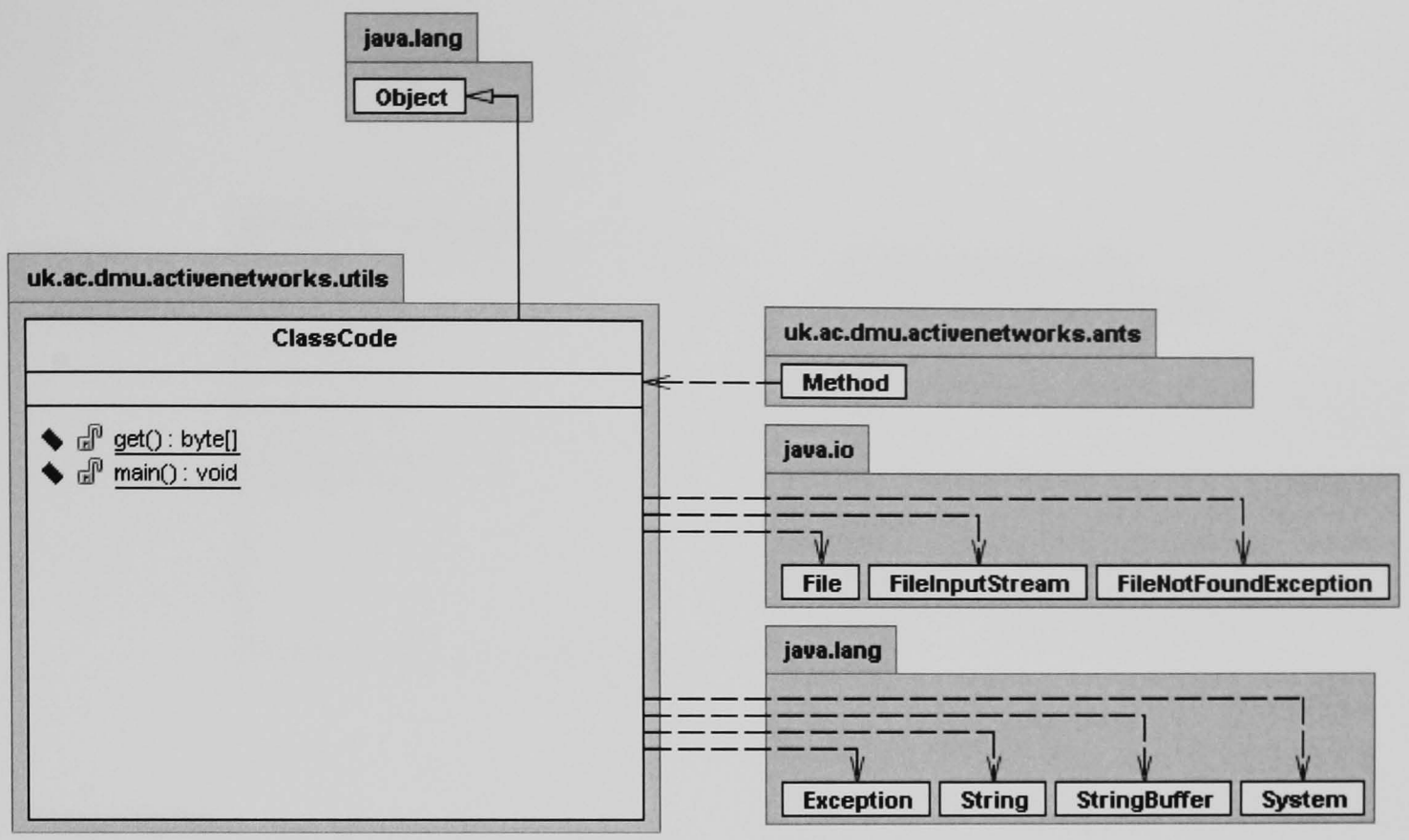
package: uk.ac.dmu.activenetworks.utils



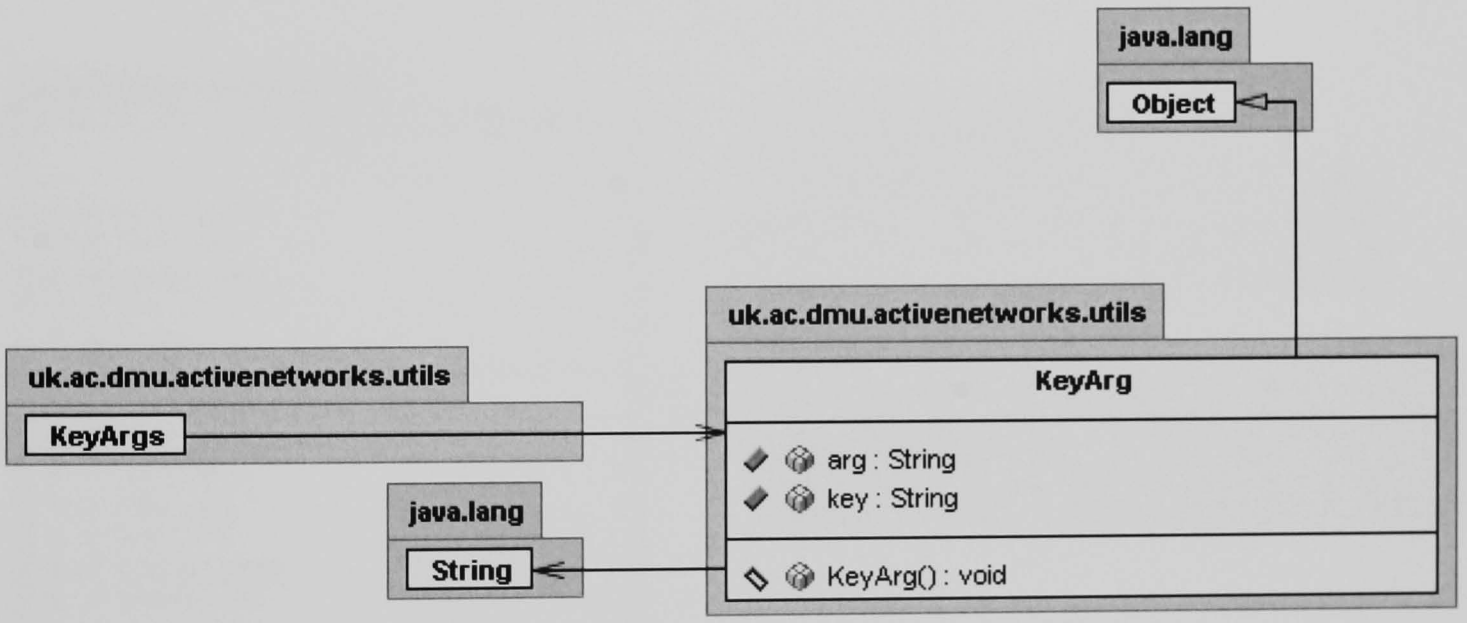
Class:ApplicationException



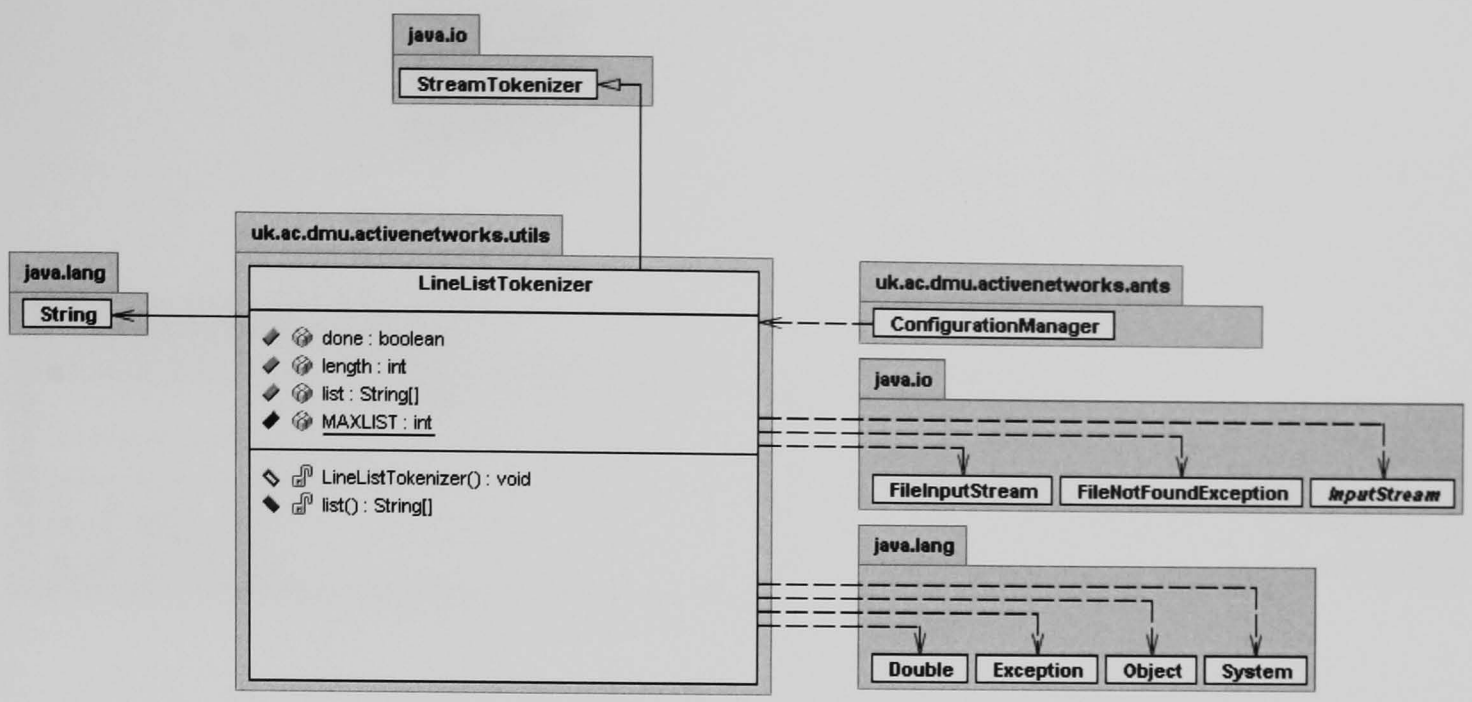
Class:ClassCode



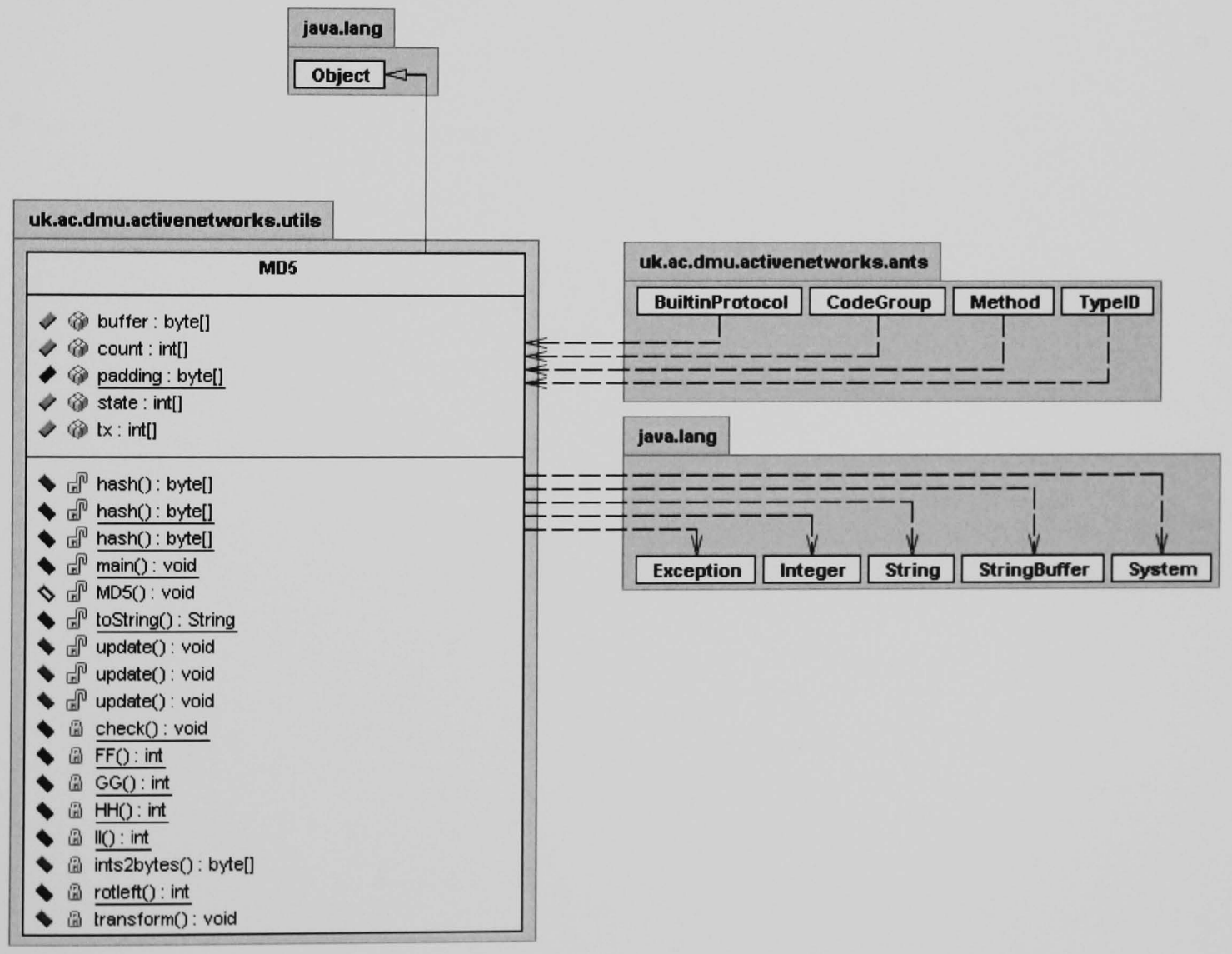
Class: KeyArg



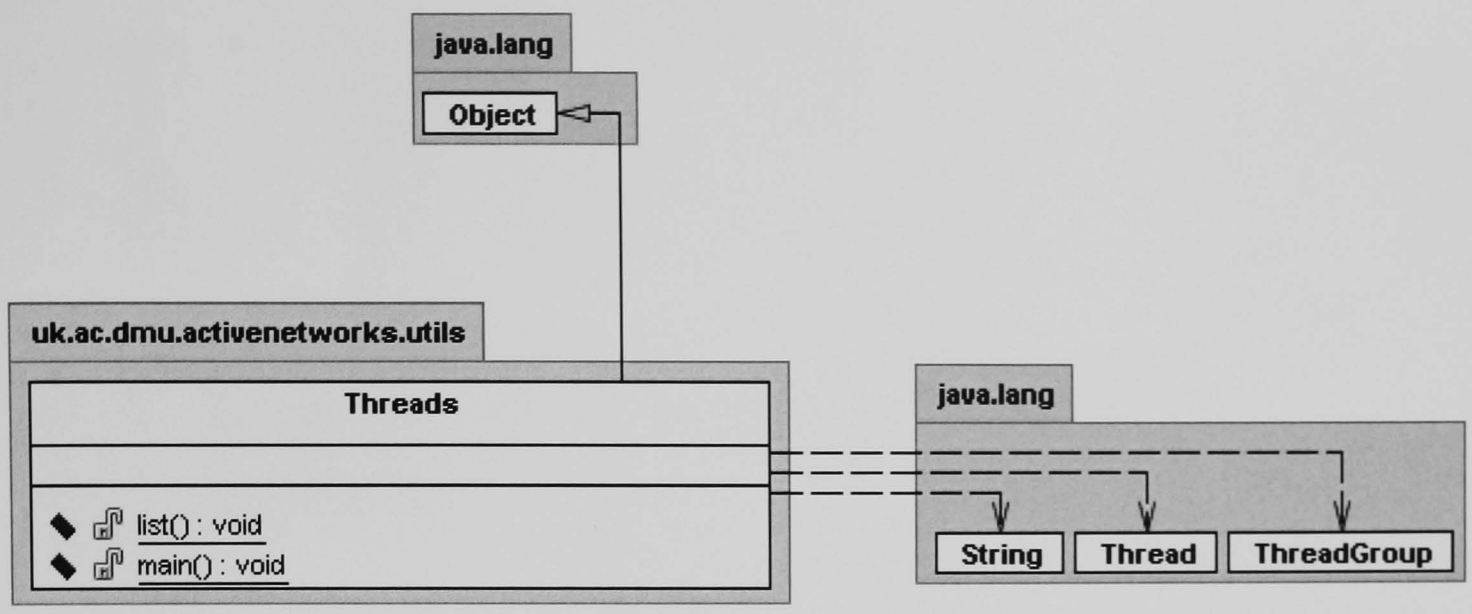
Class:LineListTokenizer



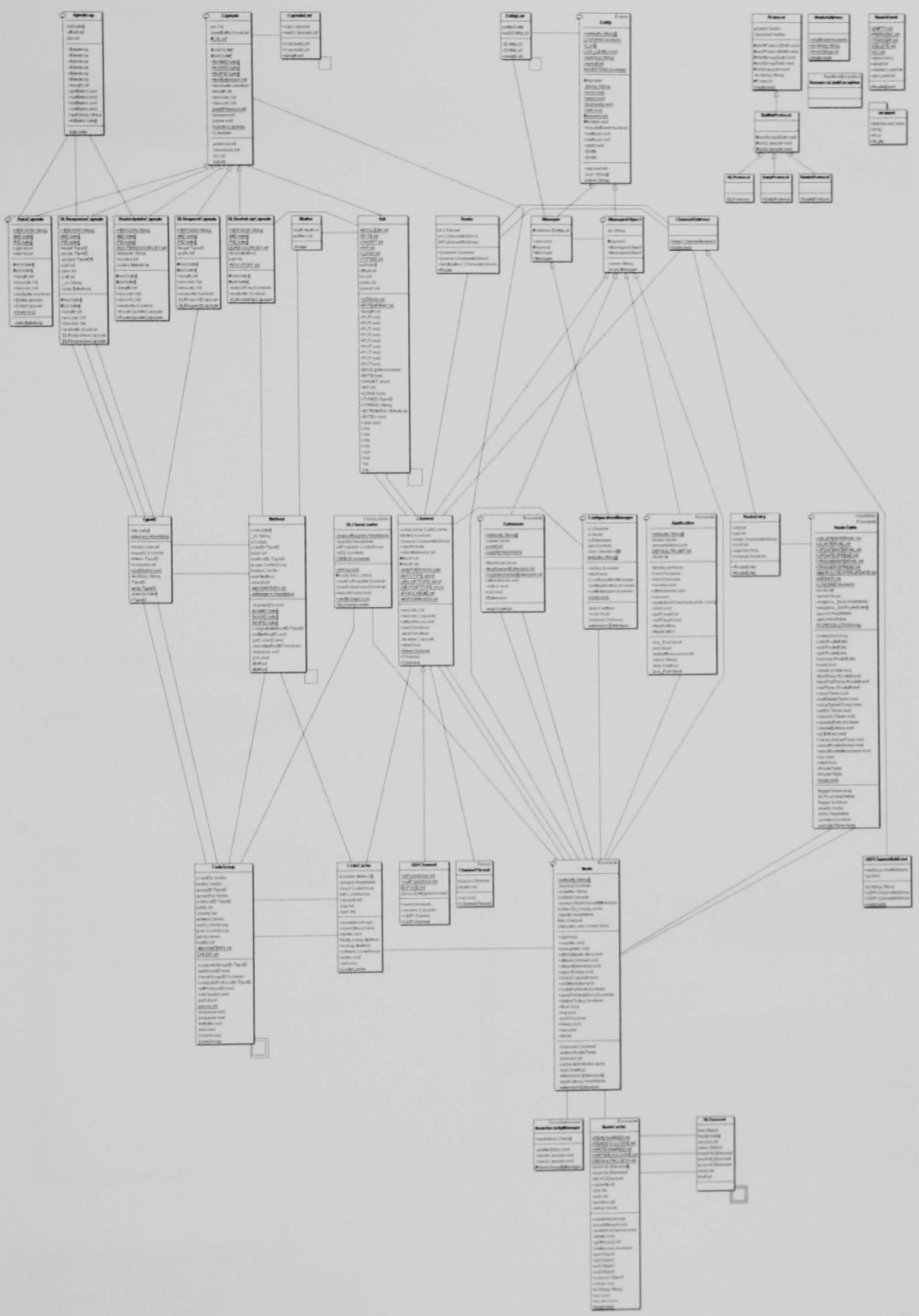
Class: MD5



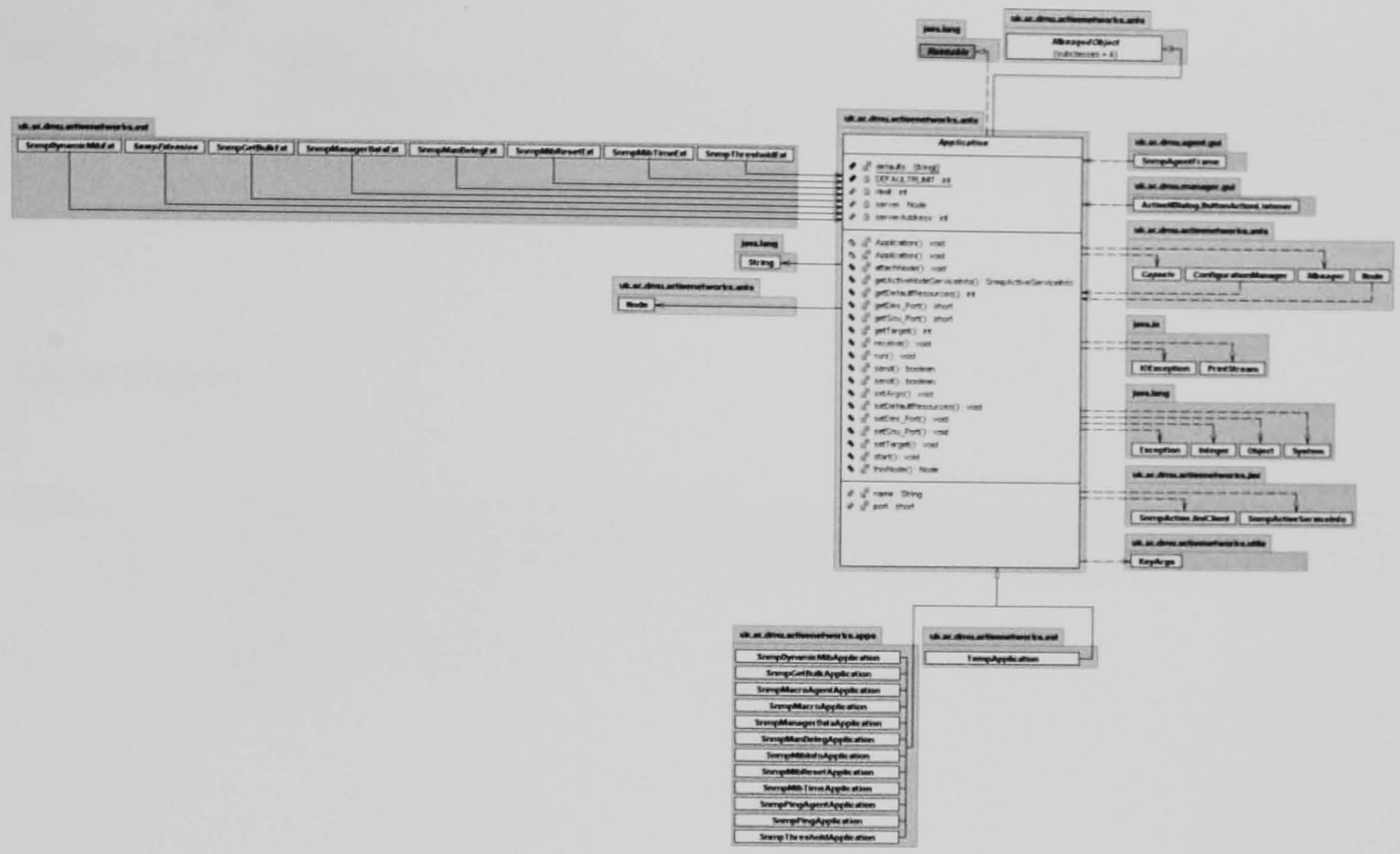
Class:Threads



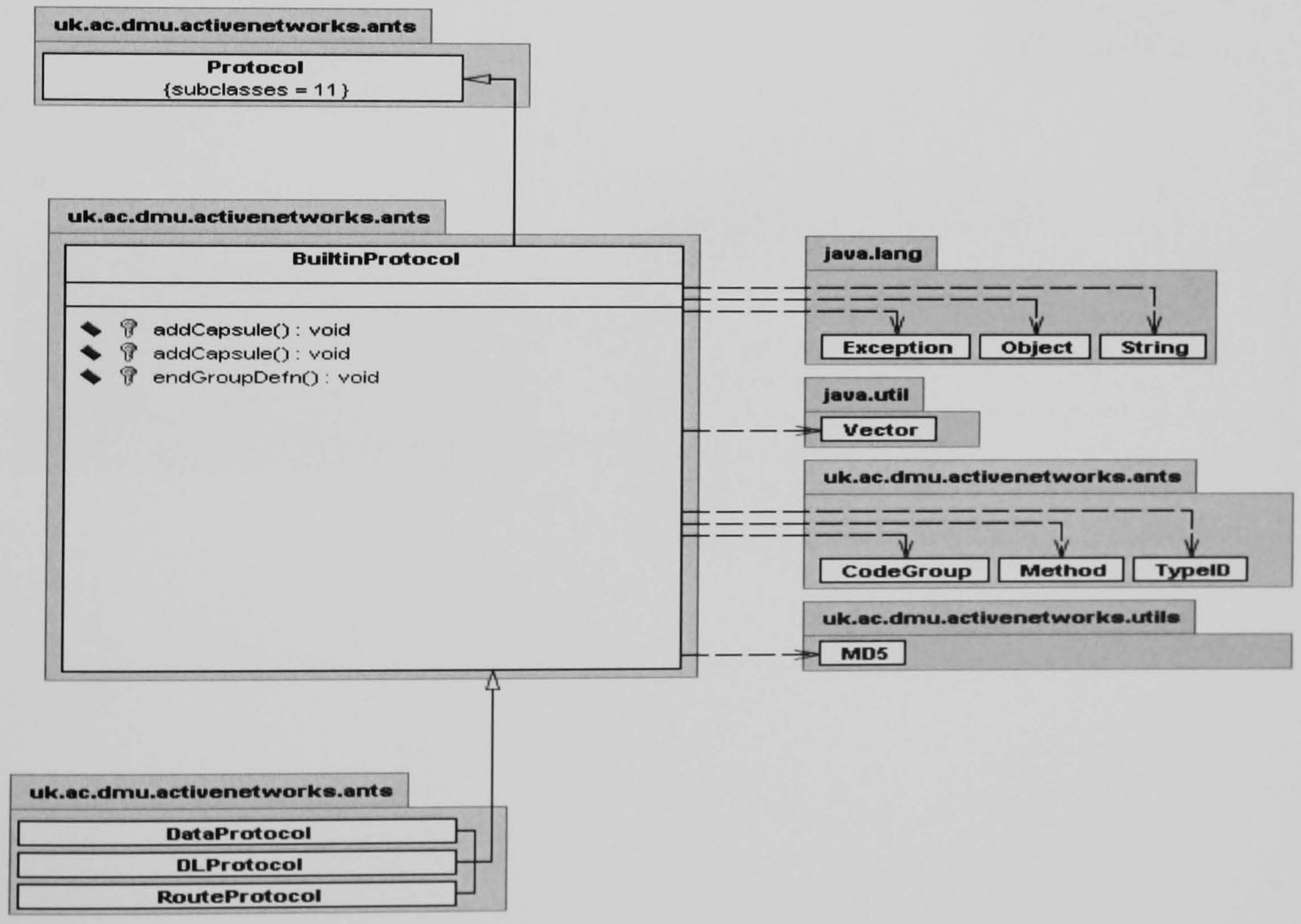
Package ac.uk.dmu.activenetworks.ants



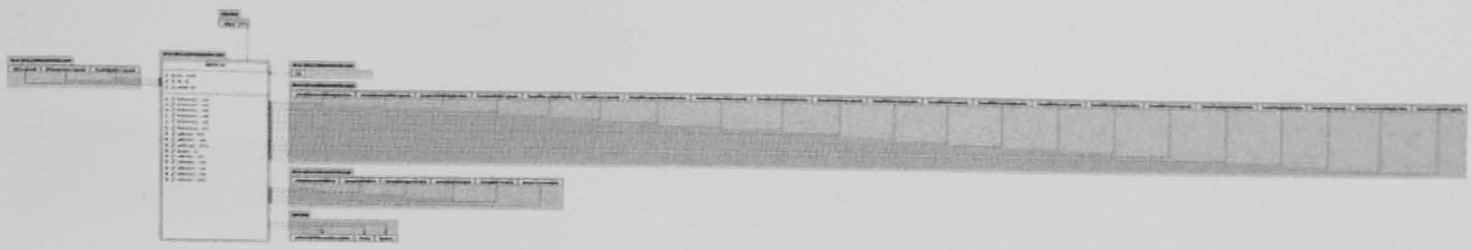
Class: Application



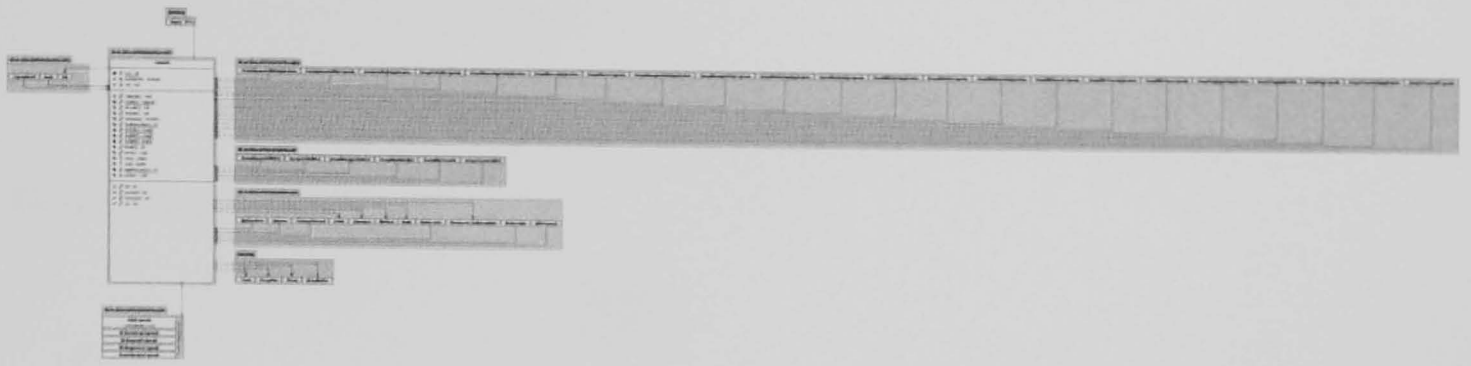
Class: BuiltinProtocol



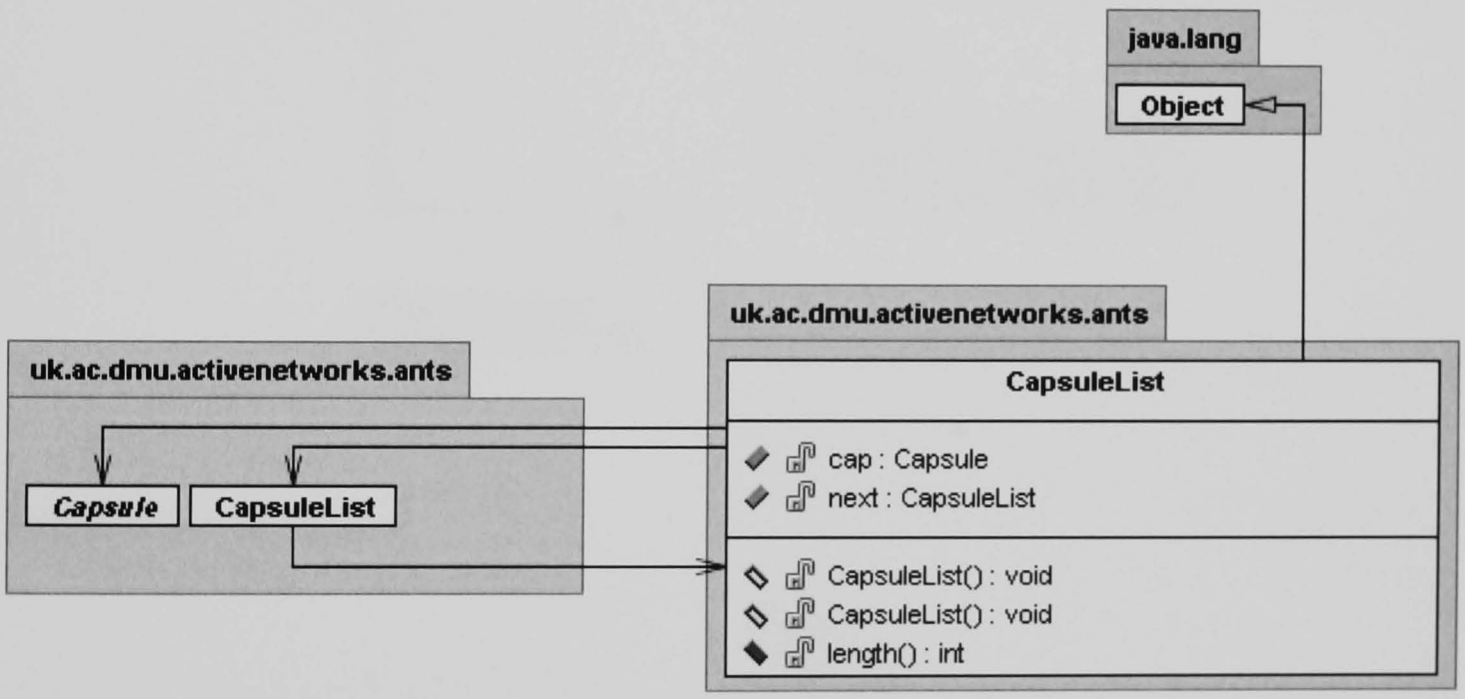
Class: ByteArray



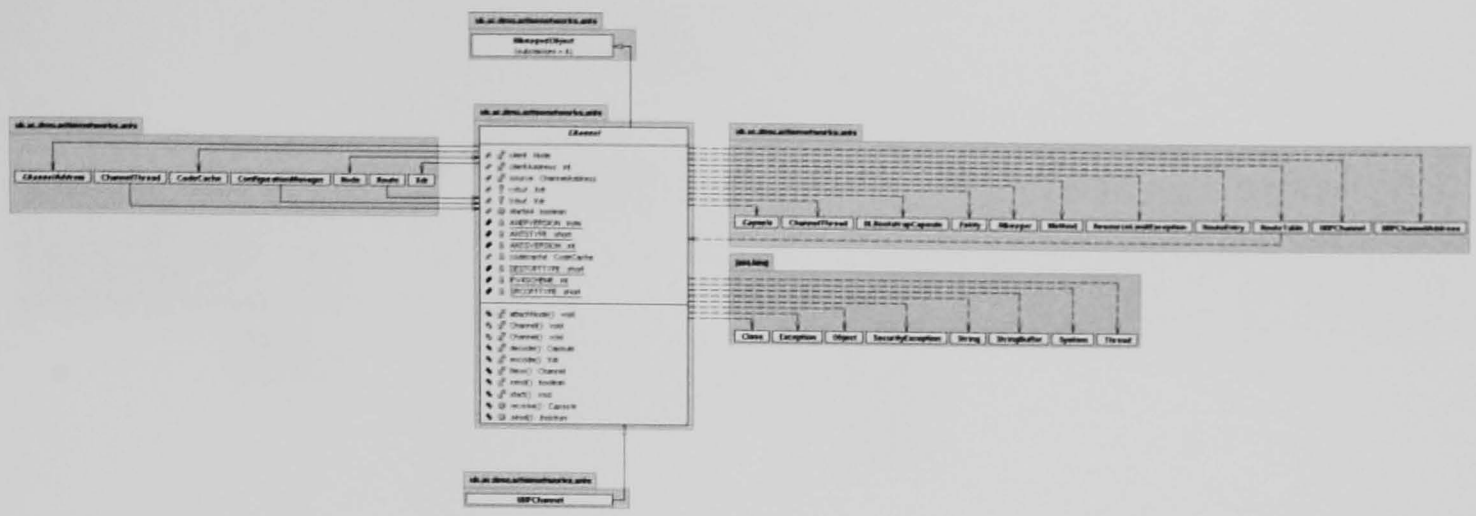
Class: Capsule



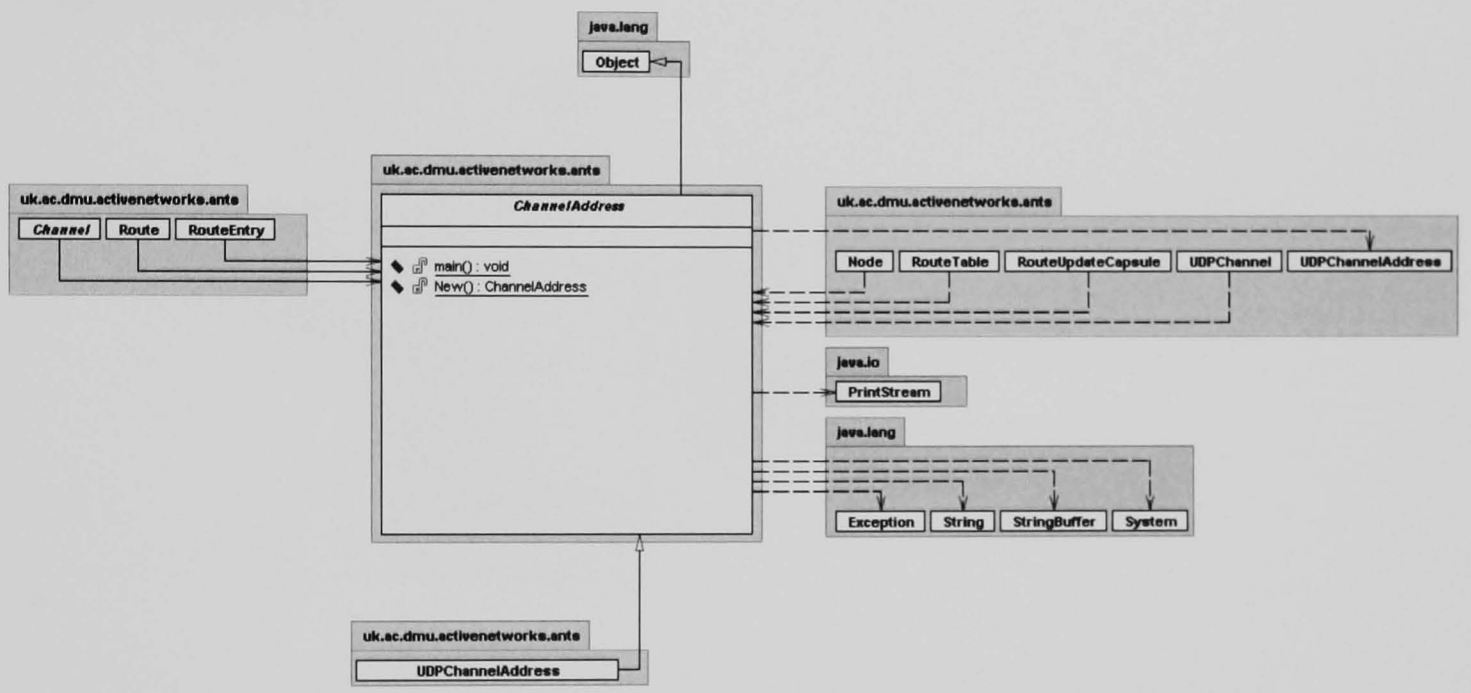
Class: CapsuleList



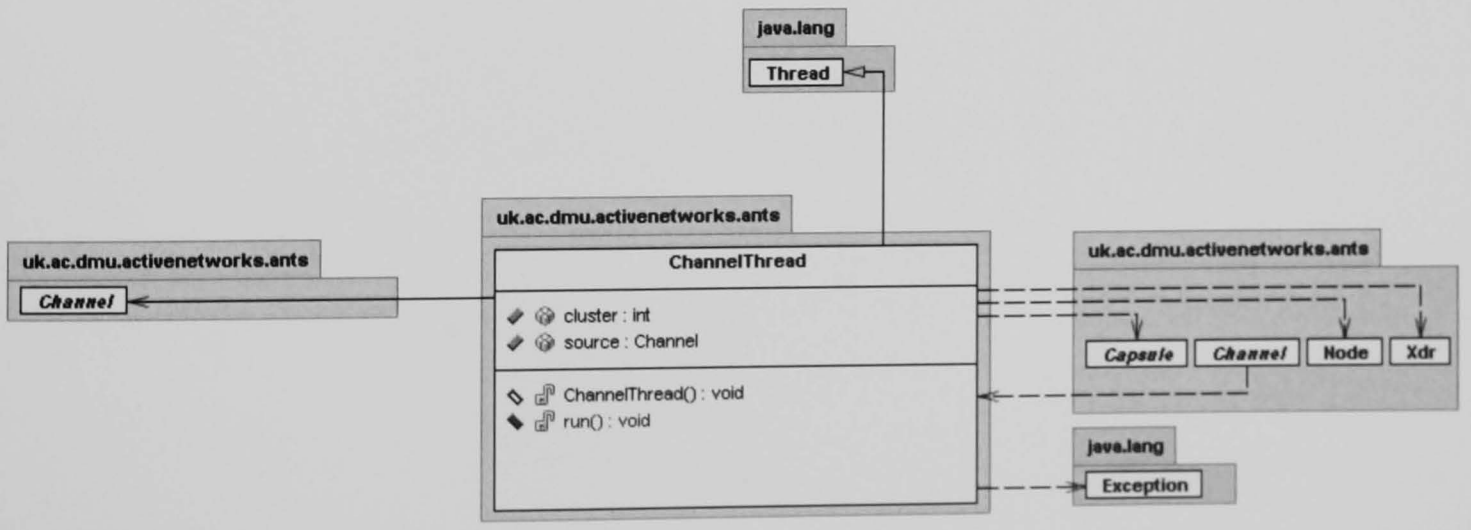
Class: Channel



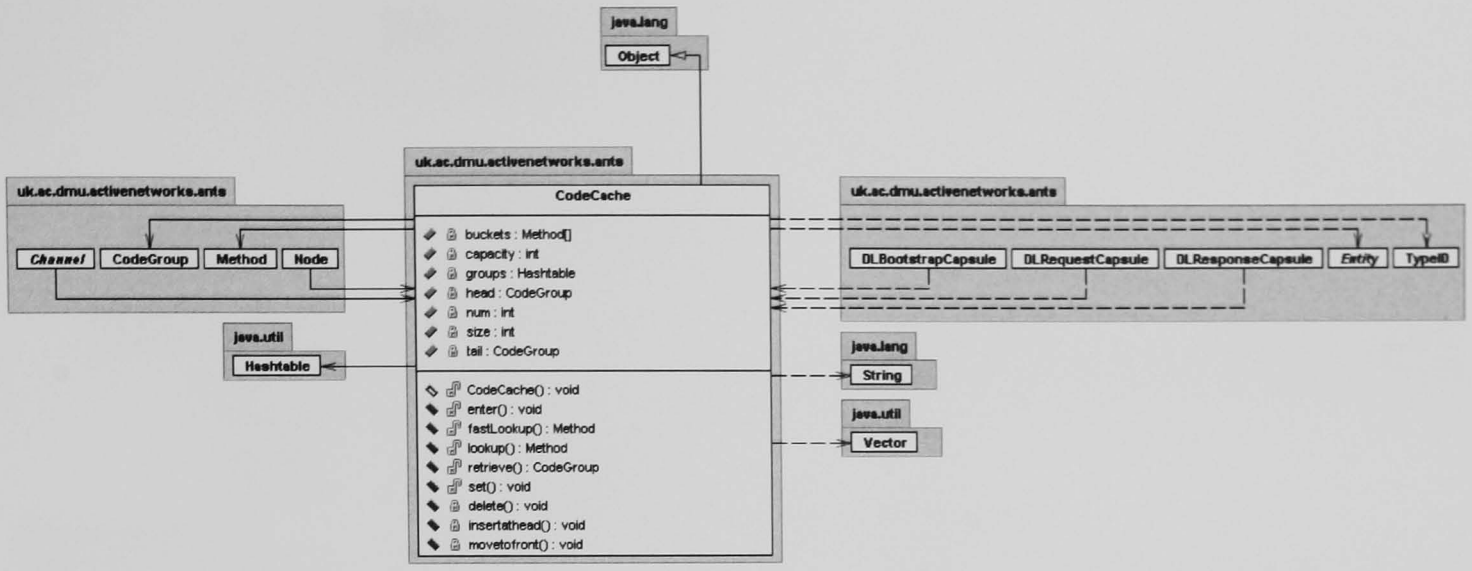
Class: ChannelAddress



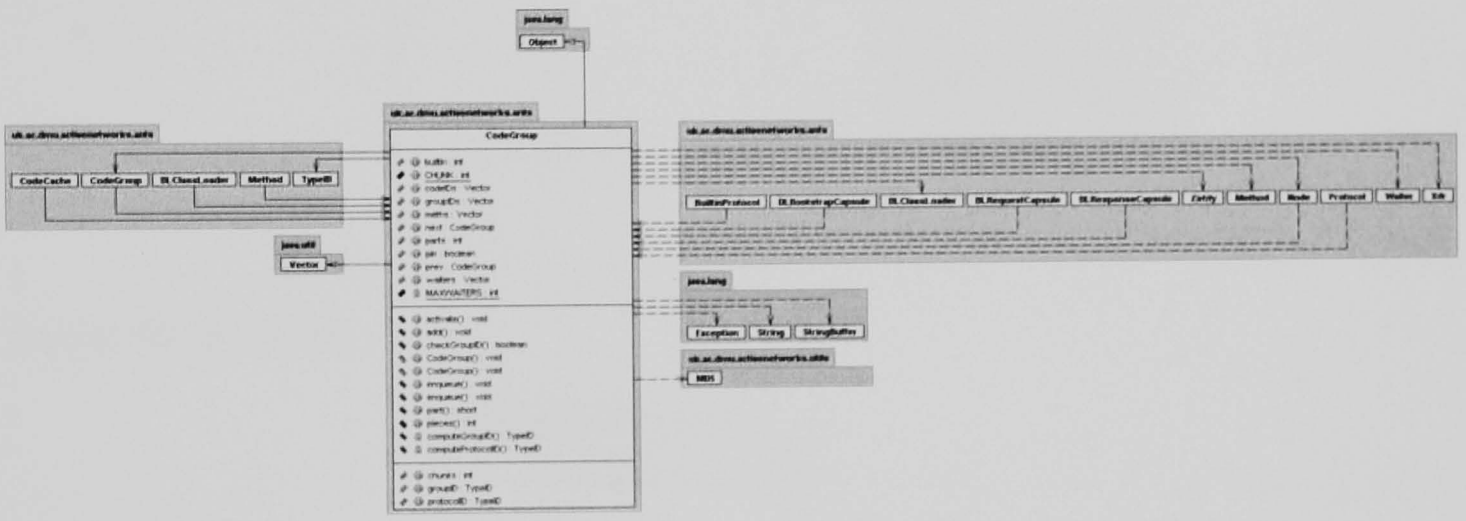
Class: ChannelThread



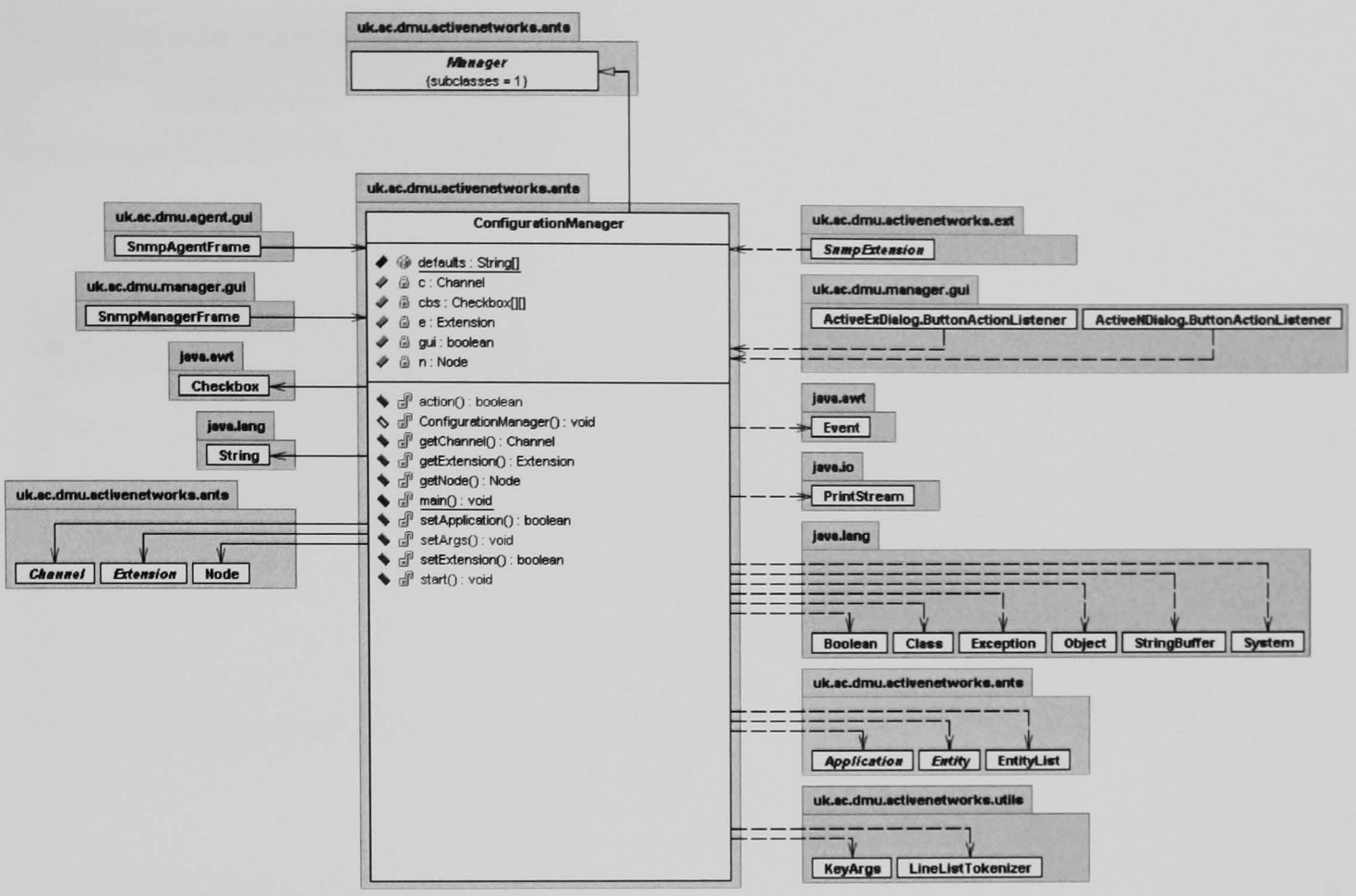
Class: CodeCache



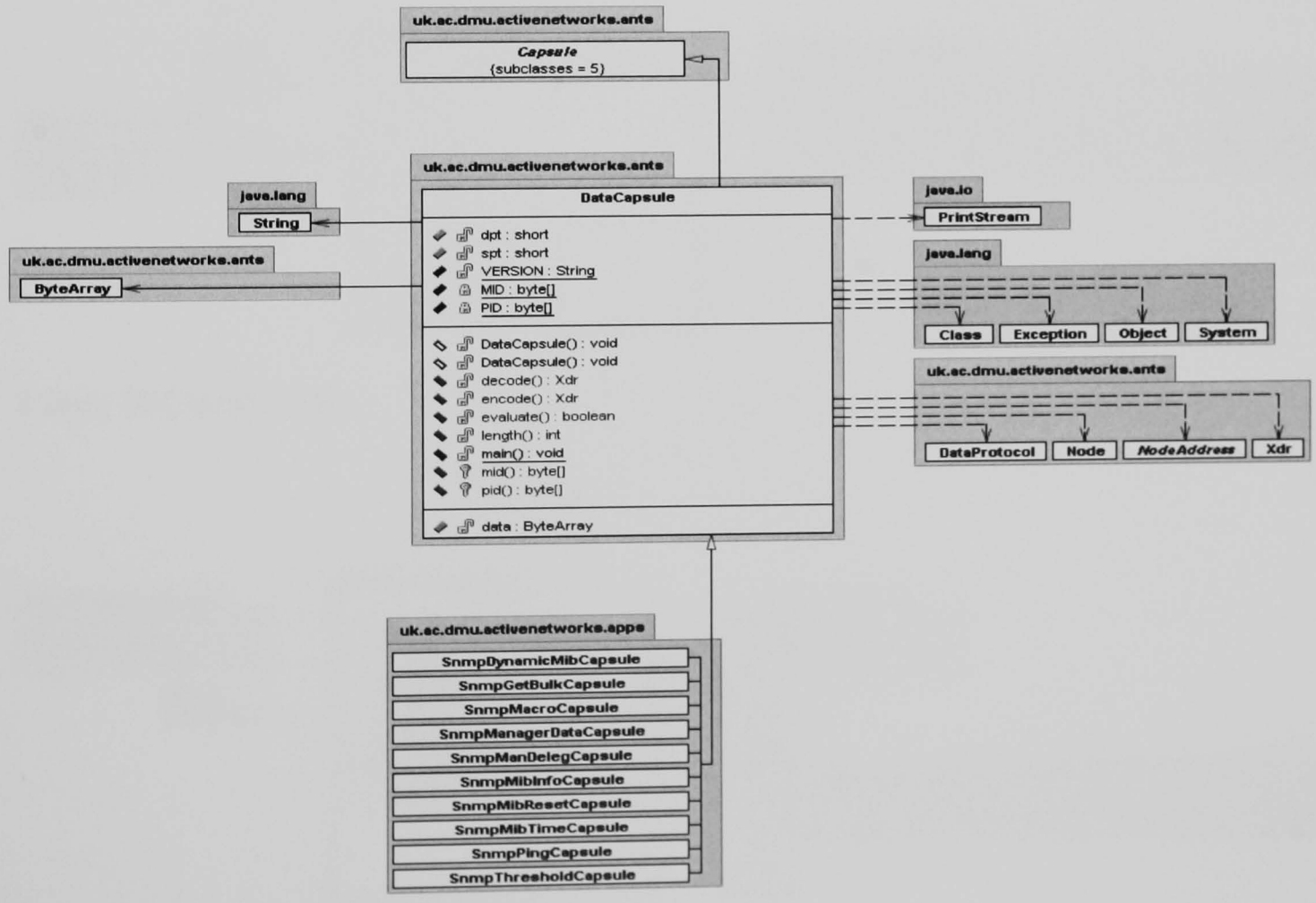
Class: CodeGroup



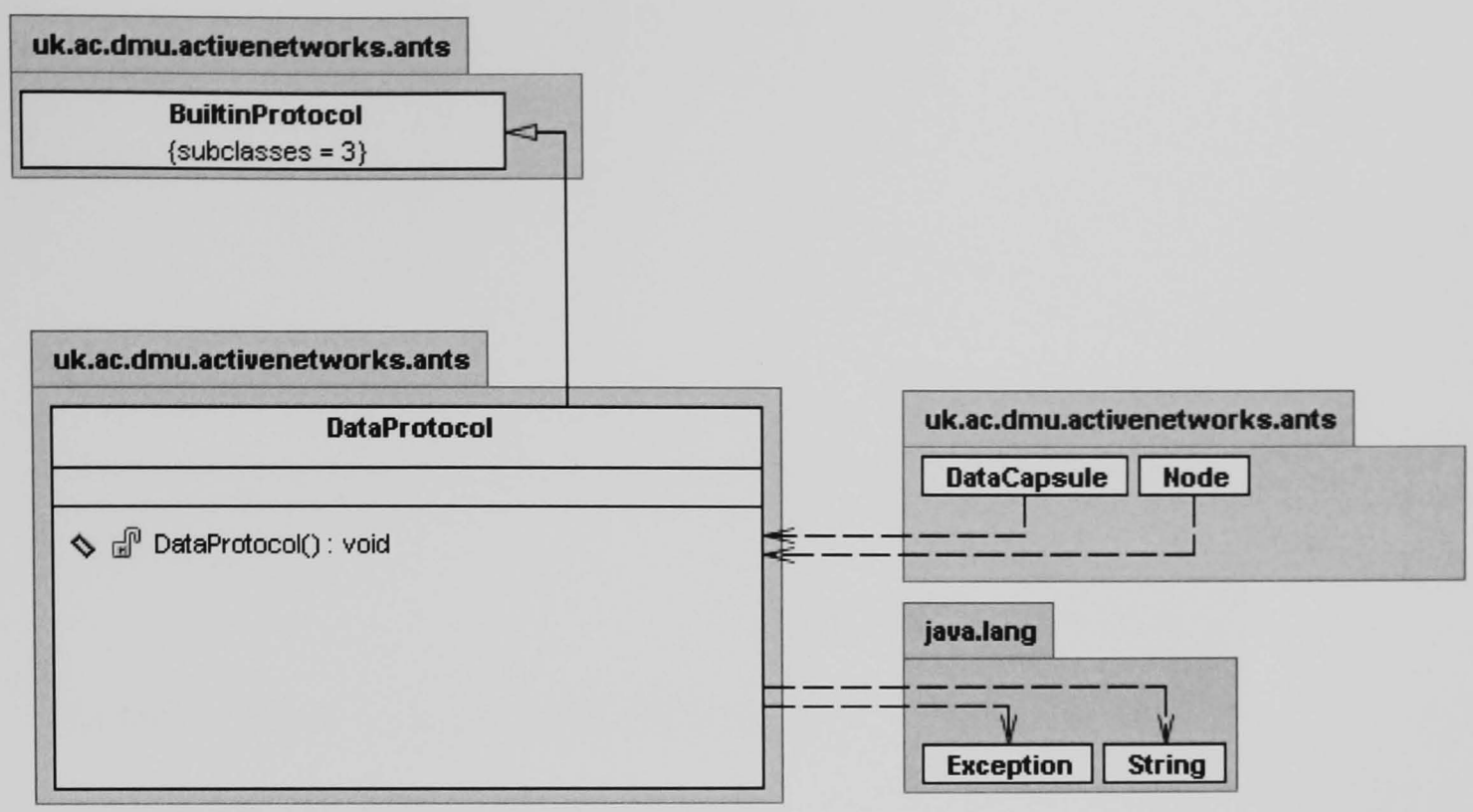
Class: ConfigurationManager



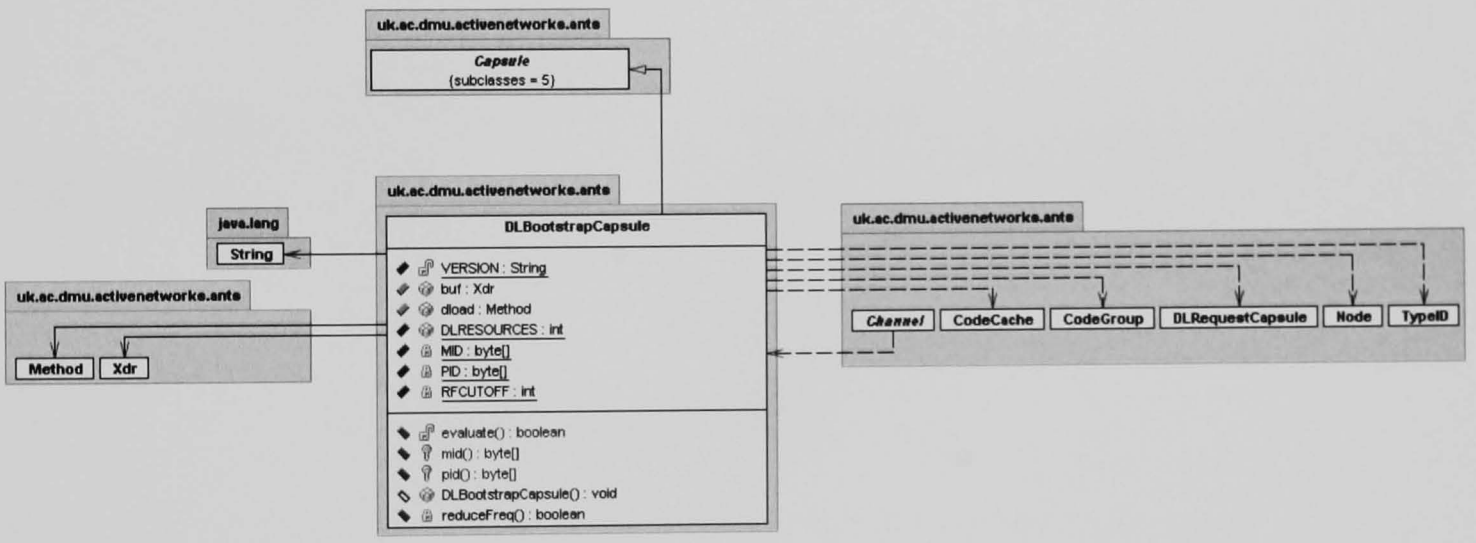
Class: DataCapsule



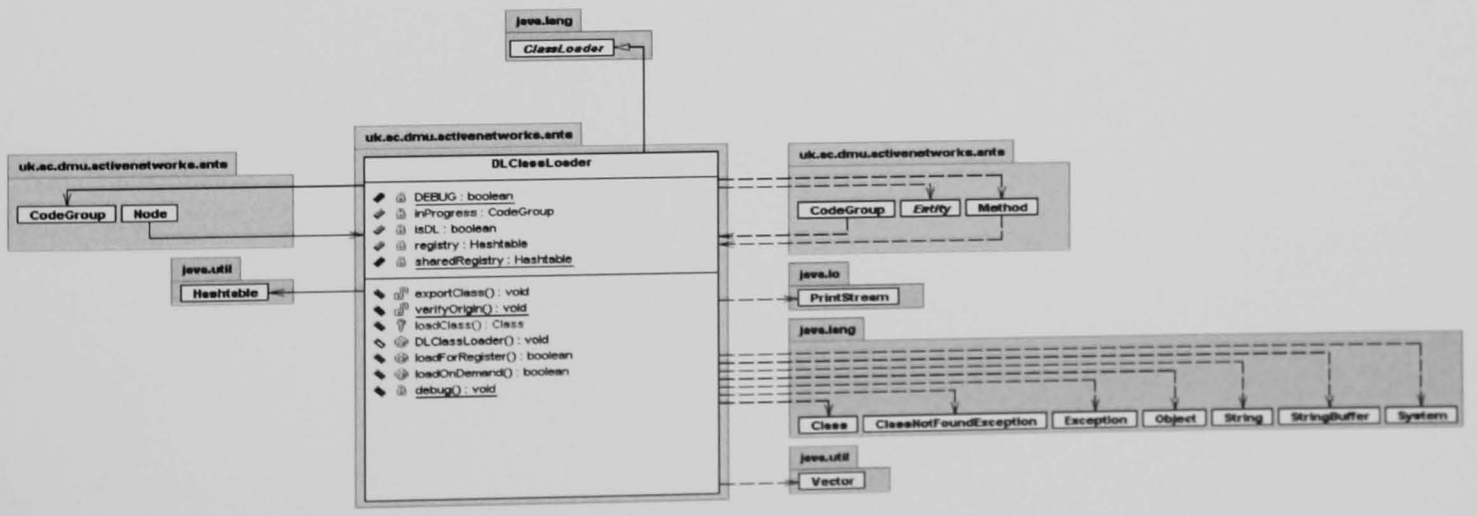
Class: DataProtocol



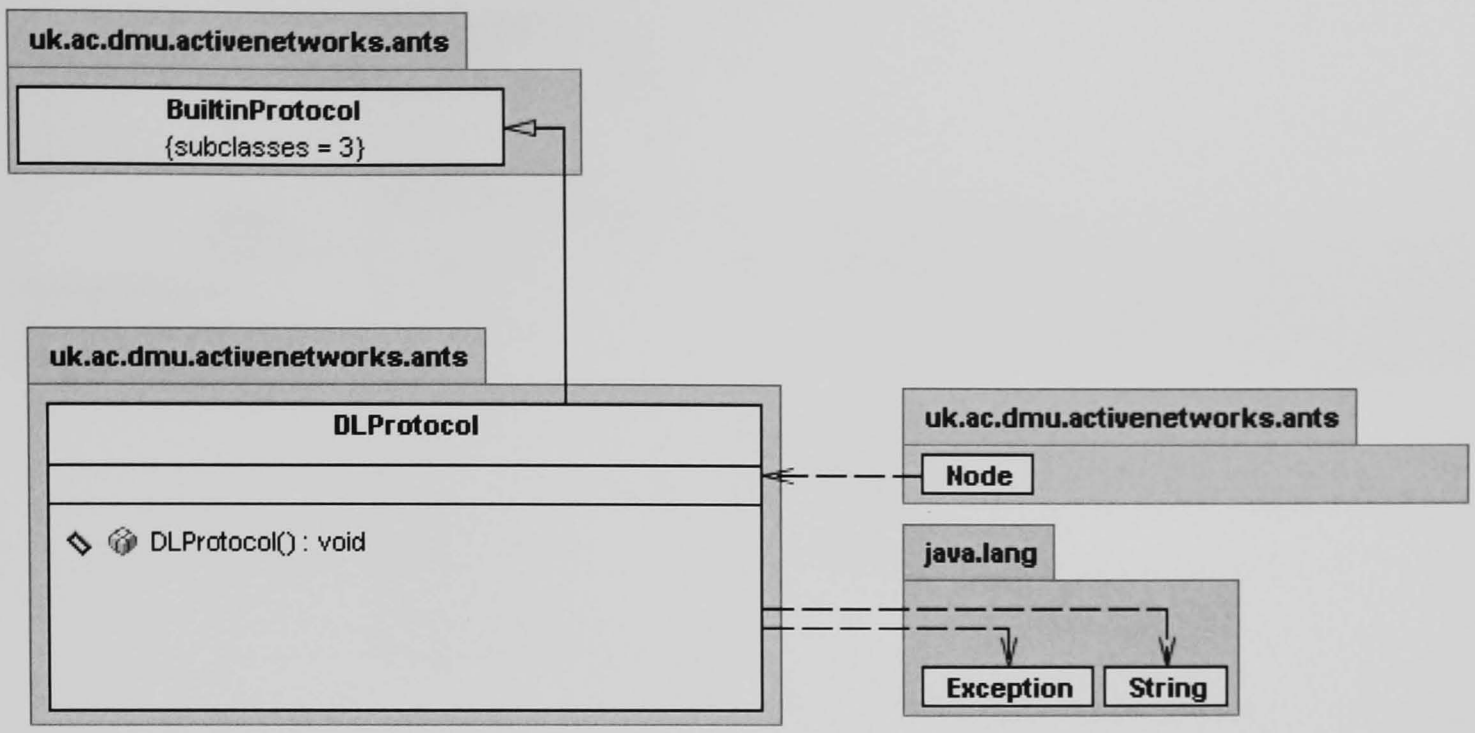
Class: DlBootstrapCapsule



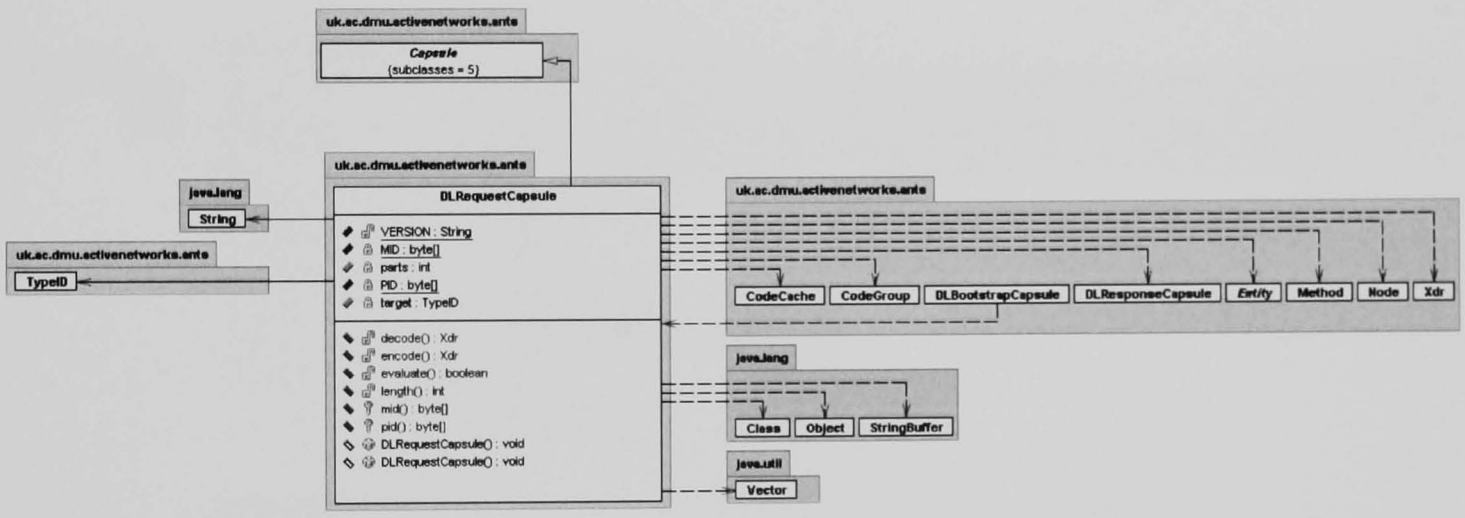
Class: DlClassLoader



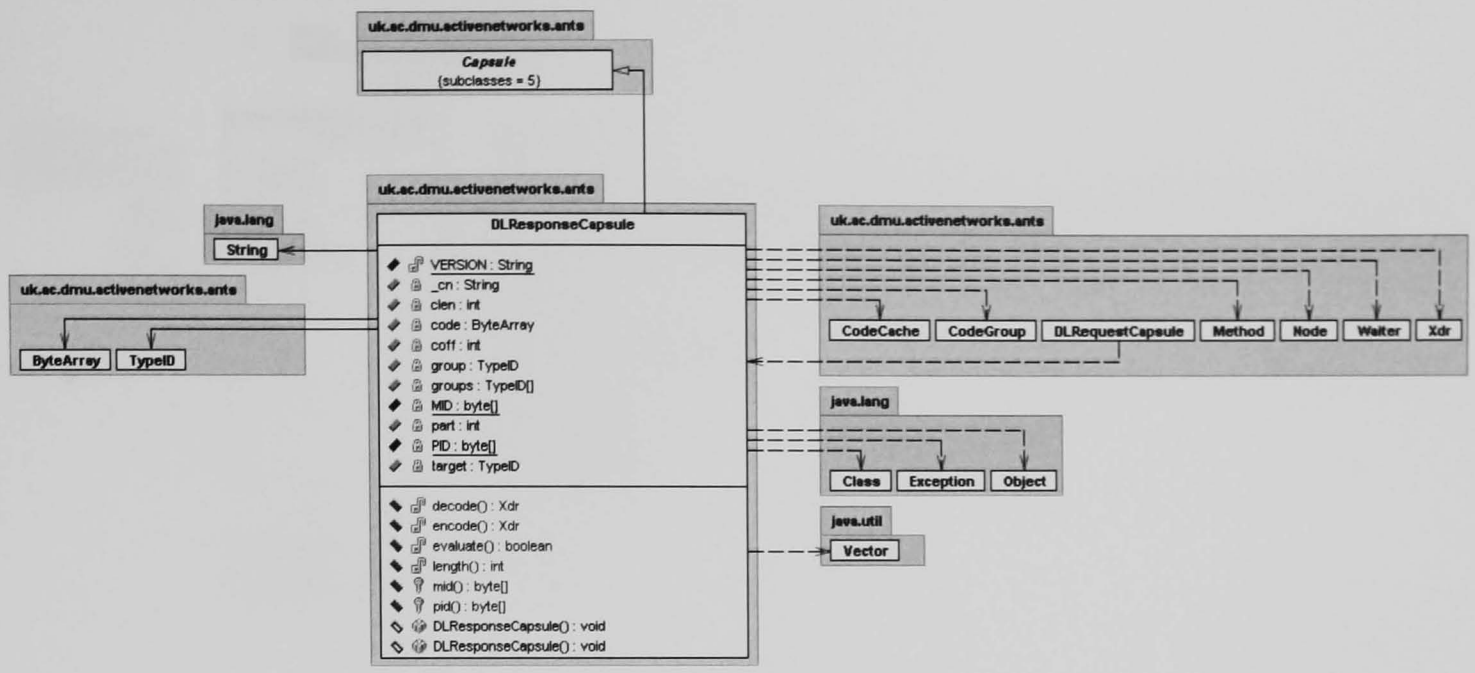
Class: DlProtocol



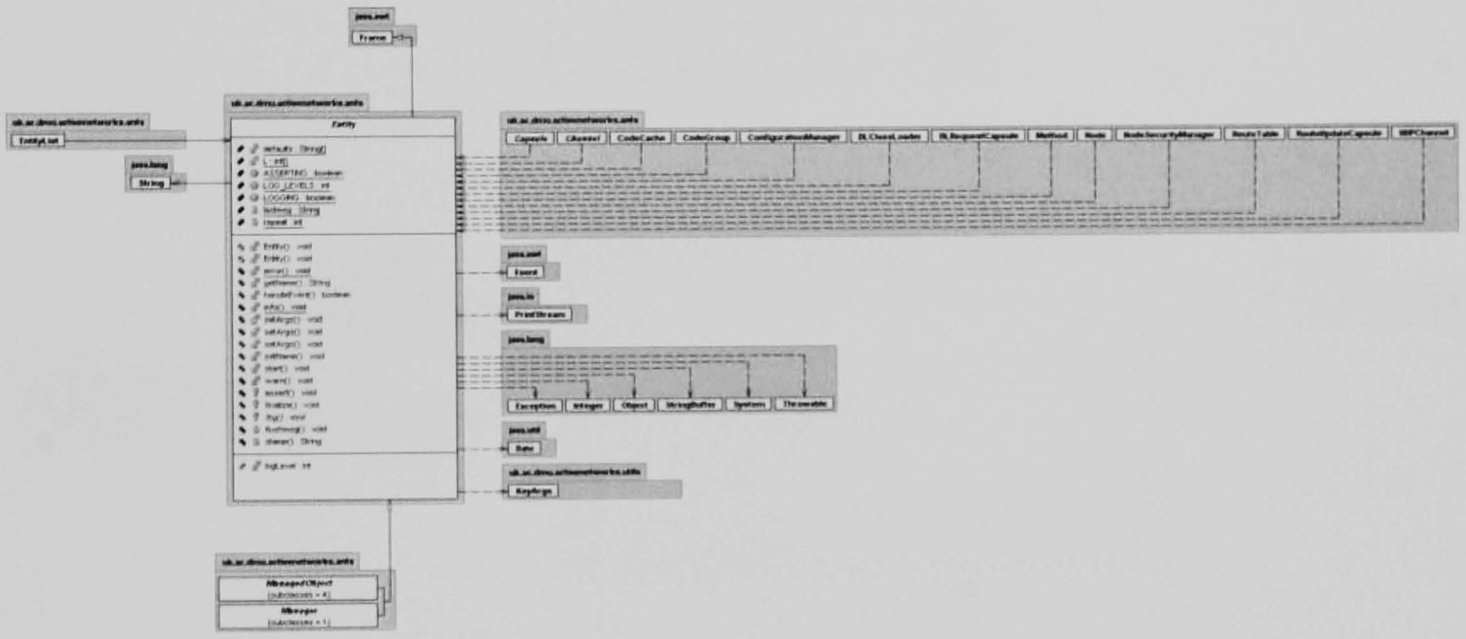
Class: DlRequestCapsule



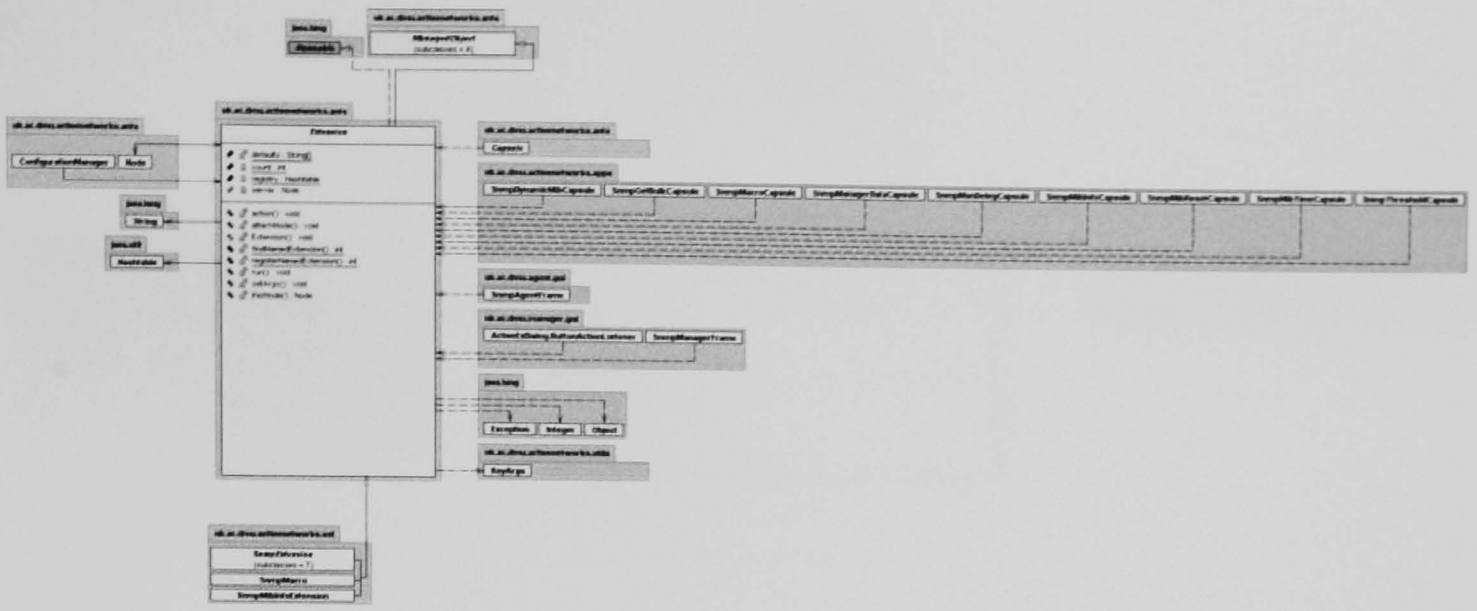
Class:DlResponseCapsule



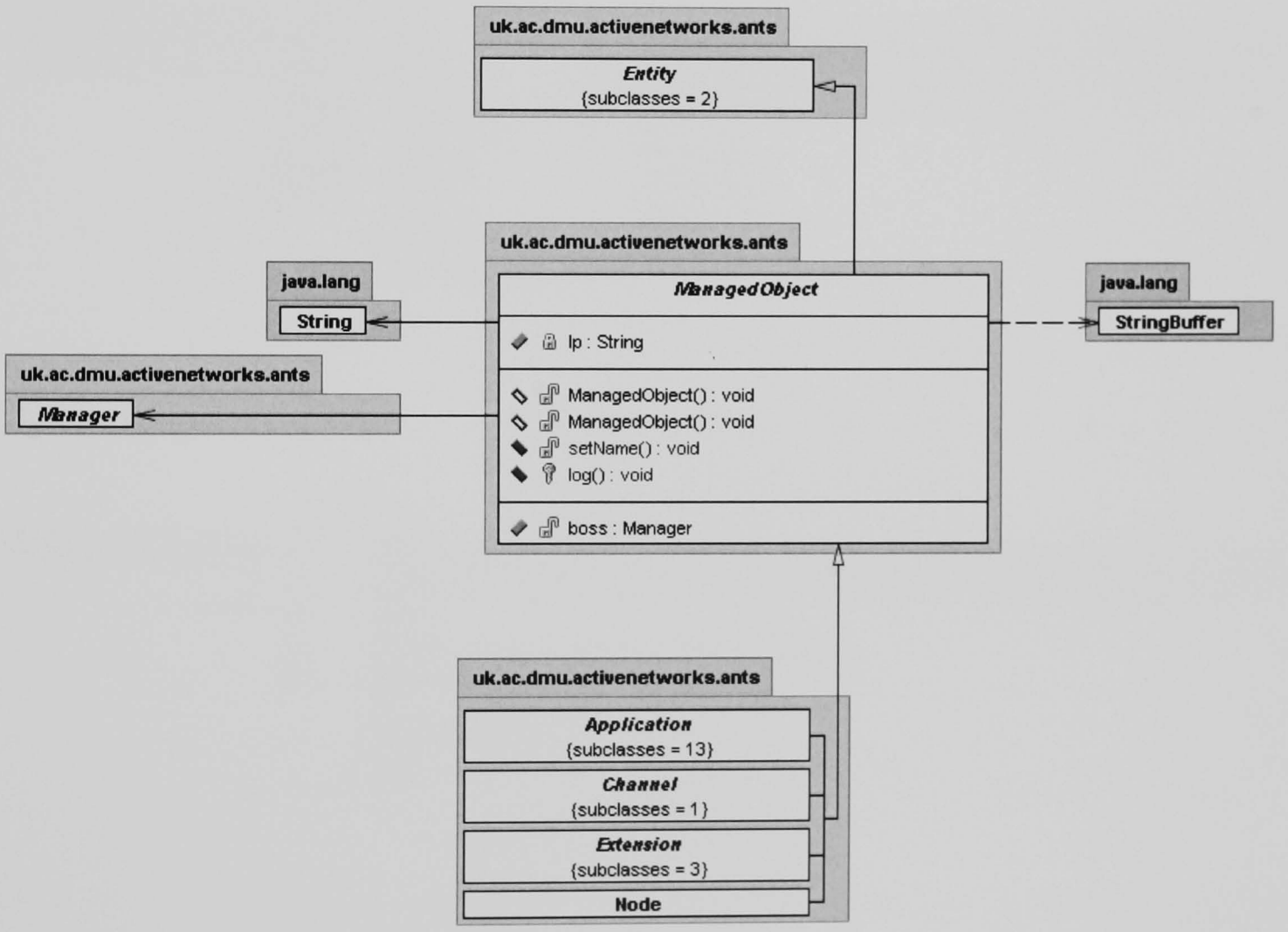
Class: Entity



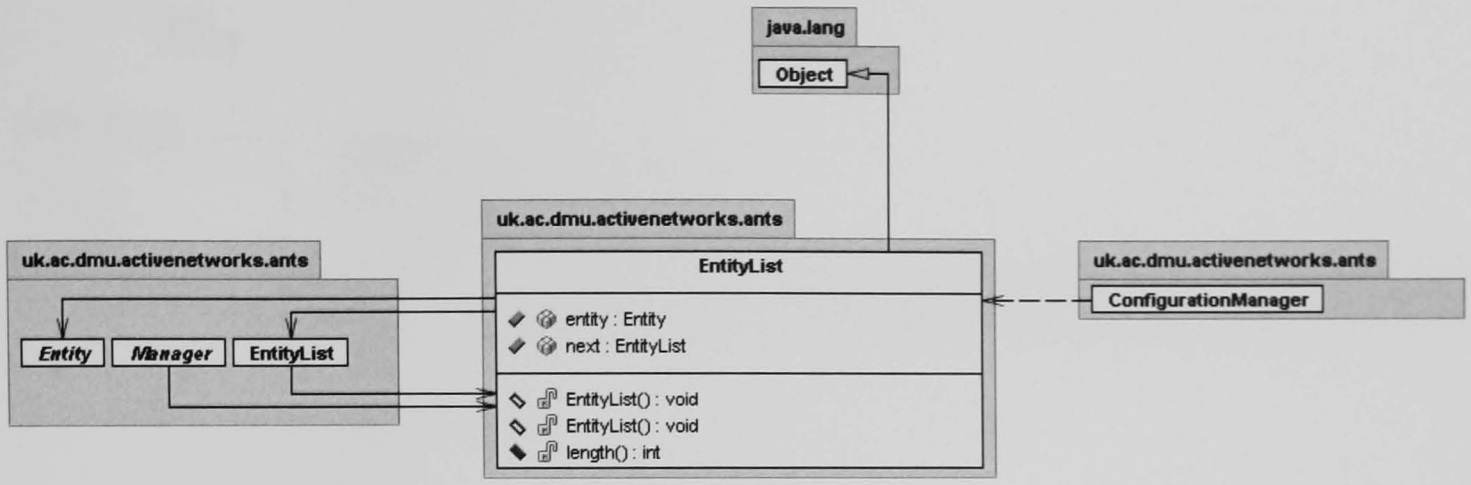
Class: Extension



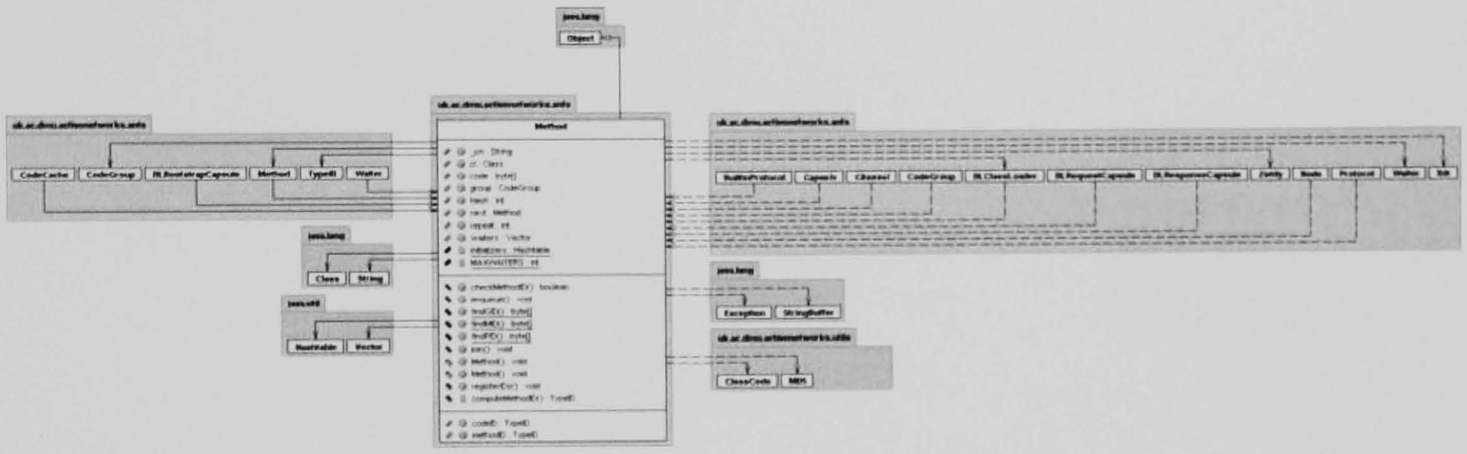
Class: ManagedObject



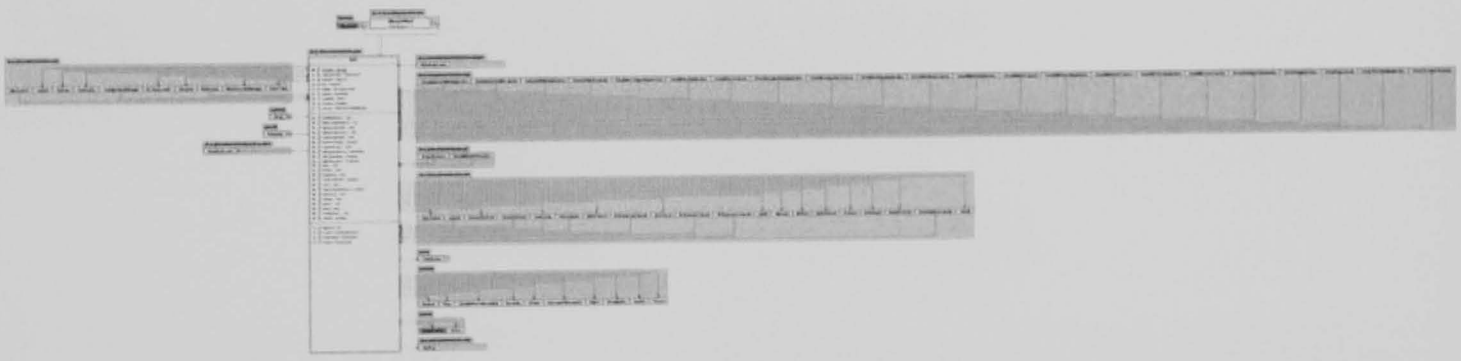
Class:Manager



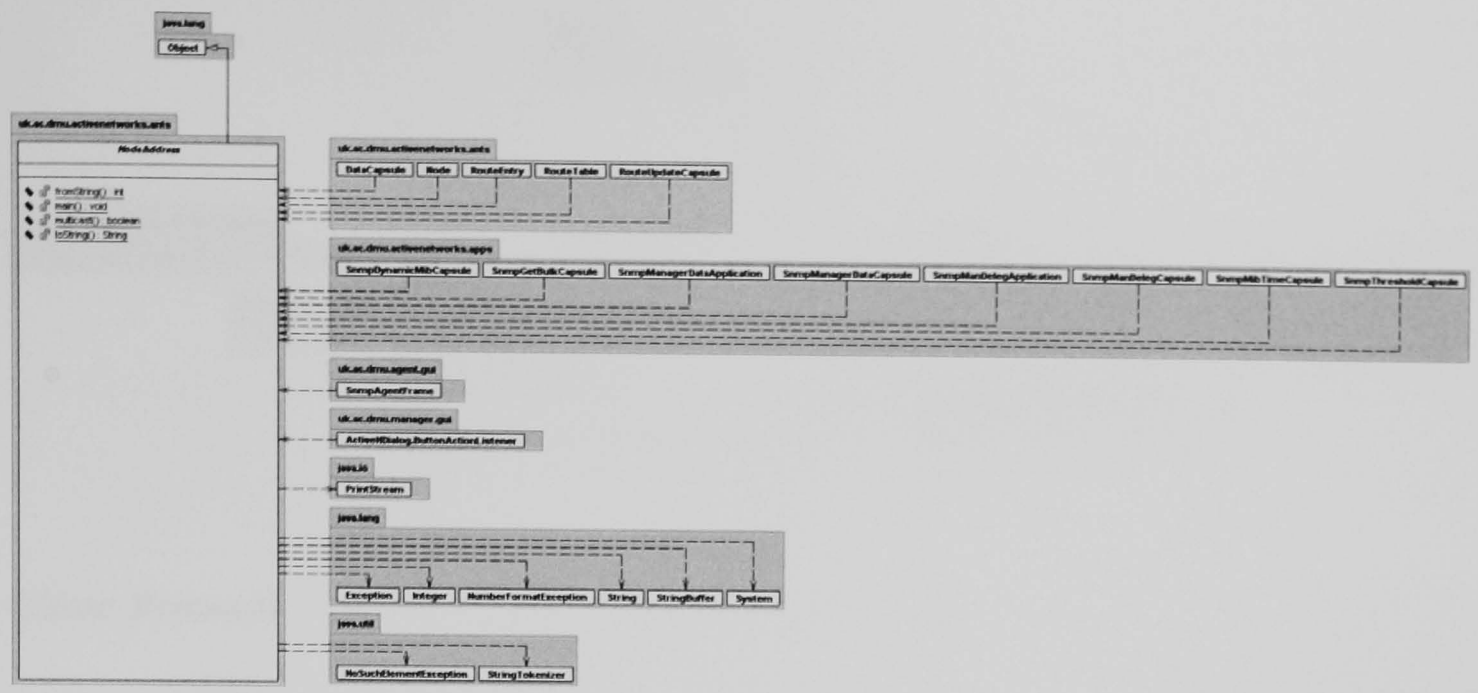
Class: Method



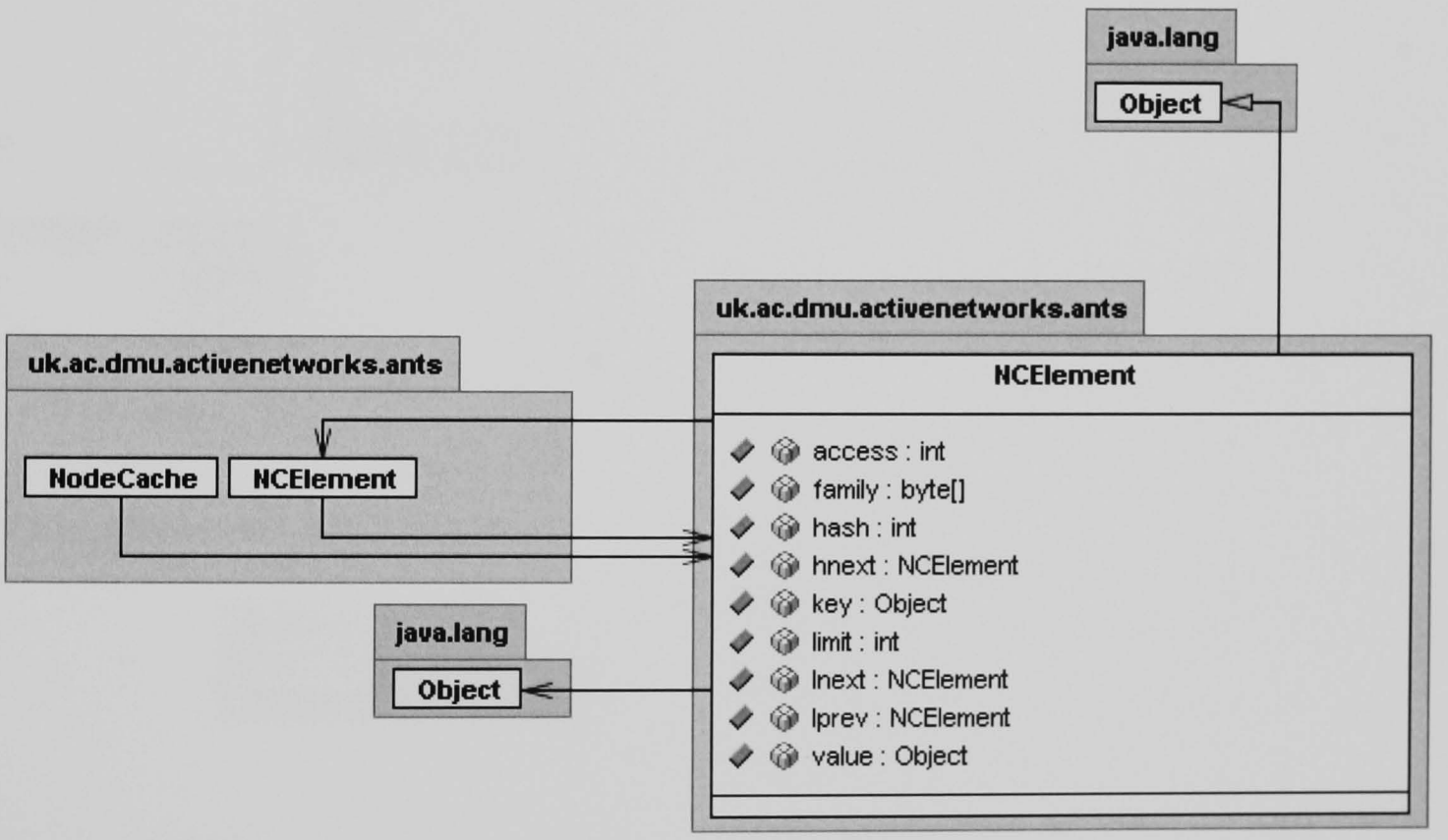
Class: Node



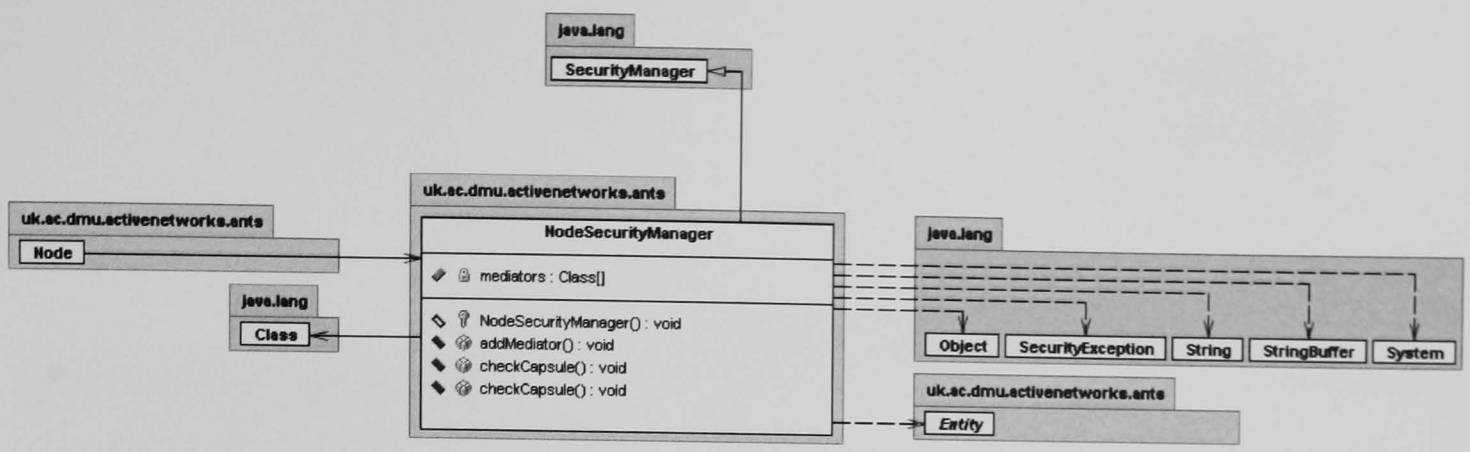
Class:NodeAddress



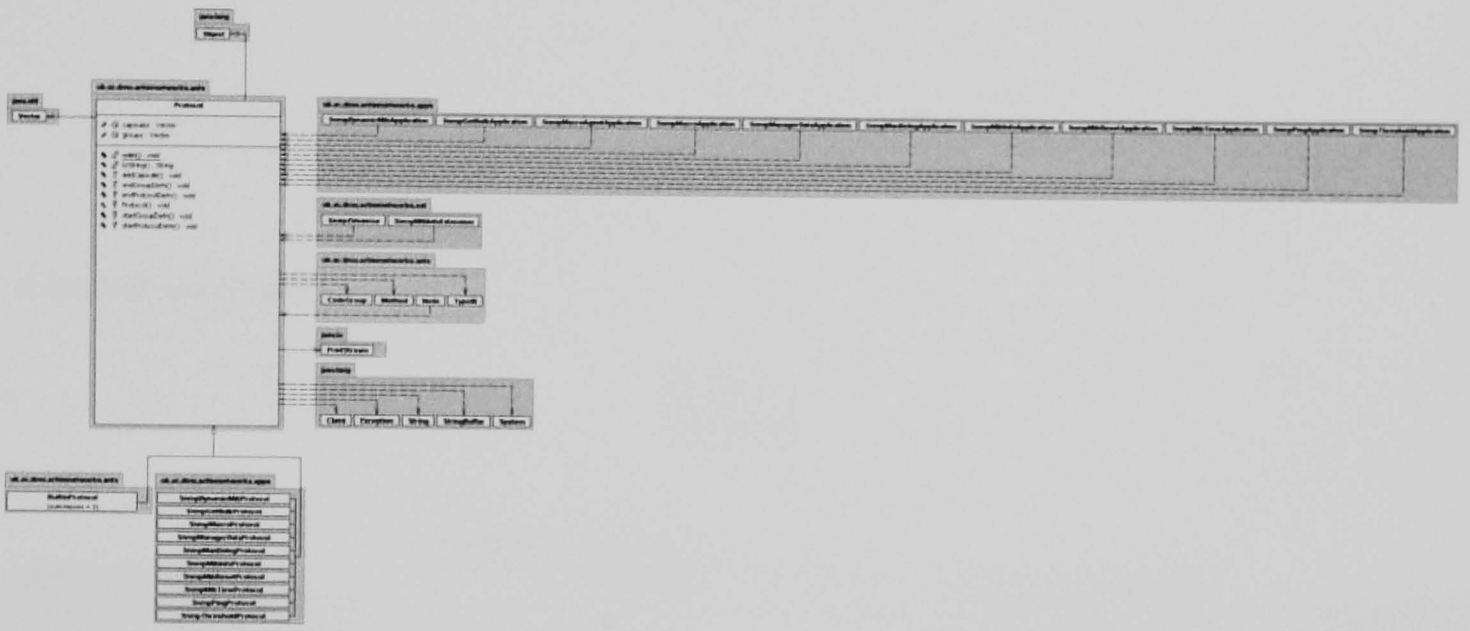
Class: NodeCache



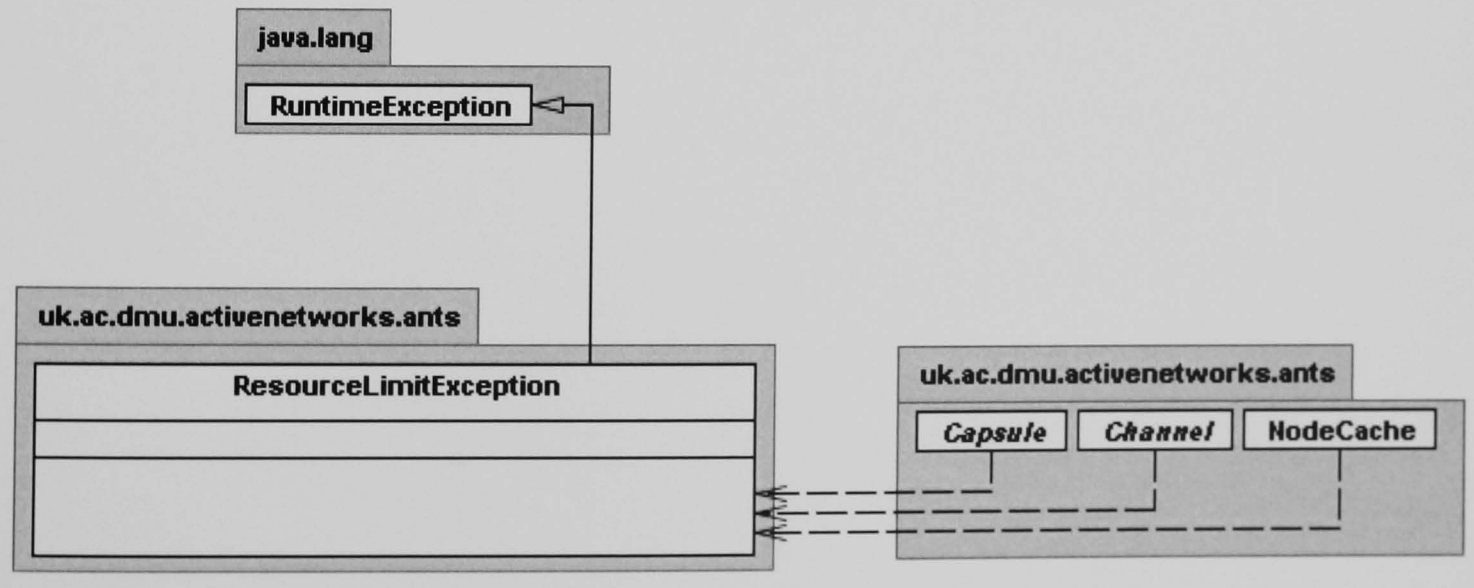
Class: NodeSecurityManager



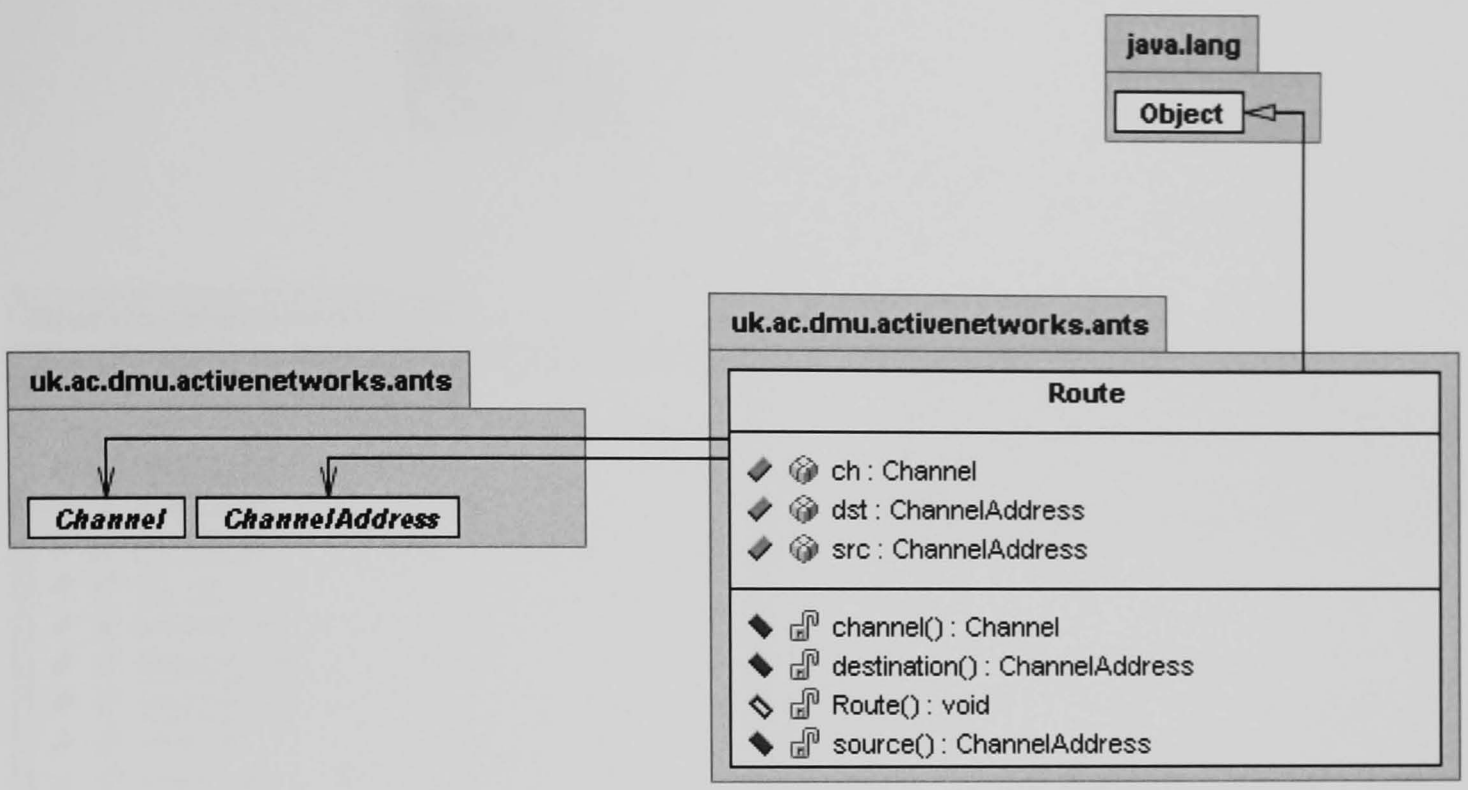
Class: Protocol



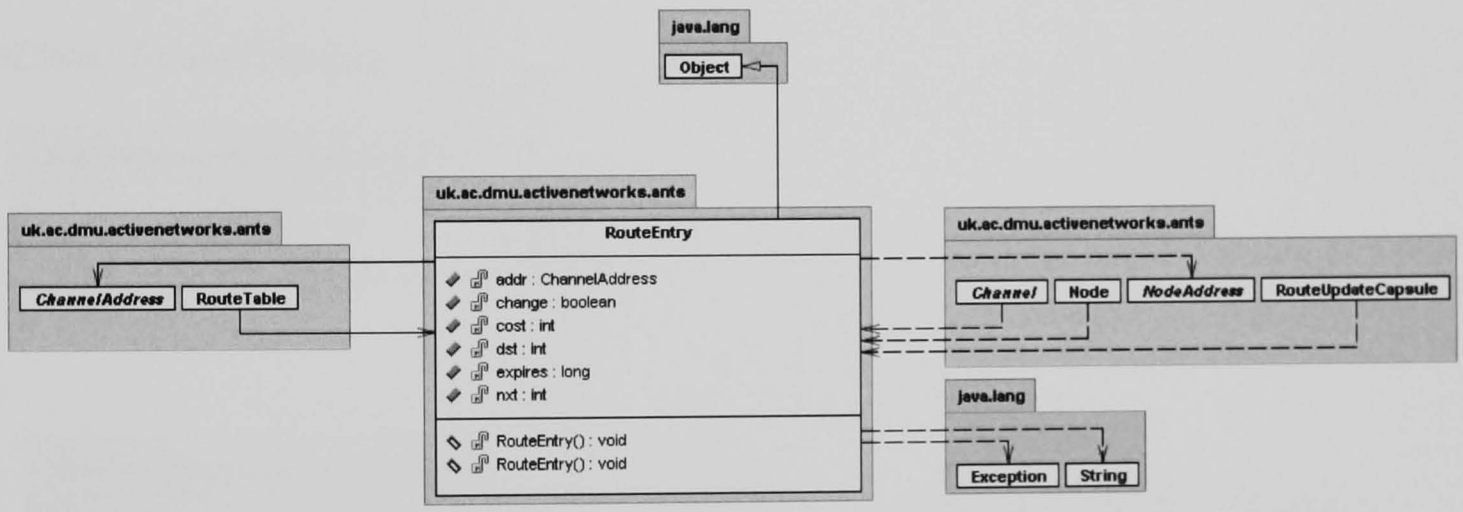
Class:ResourceLimitException



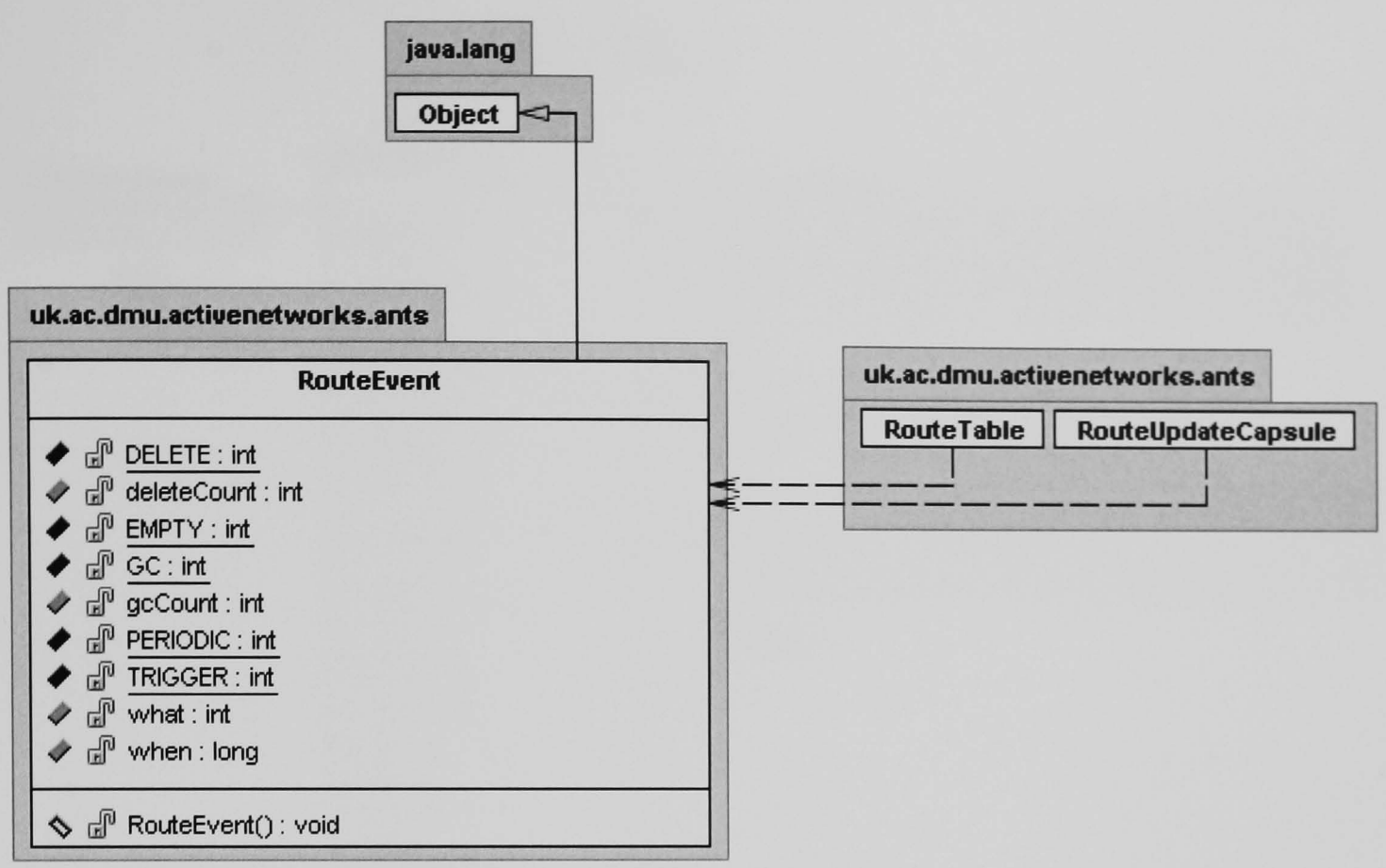
Class: Route



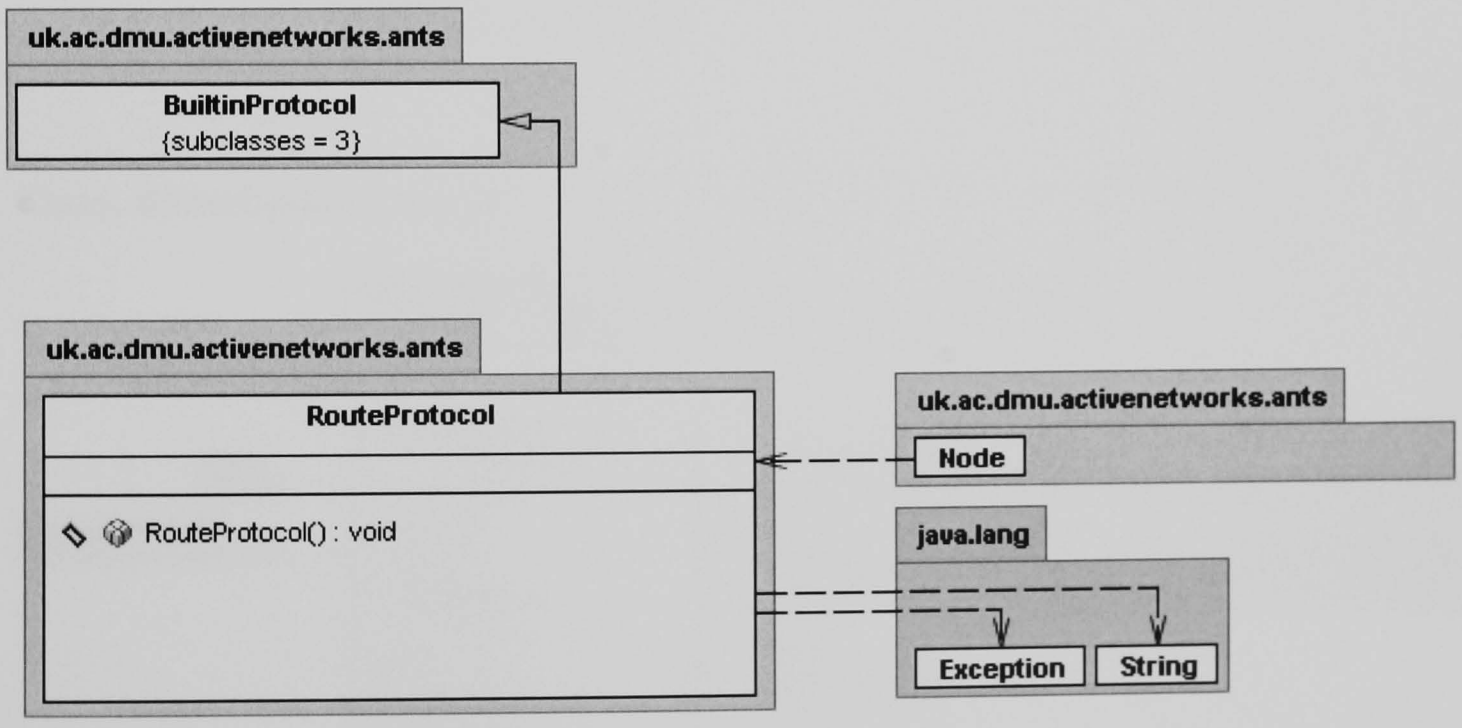
Class:RouteEntry



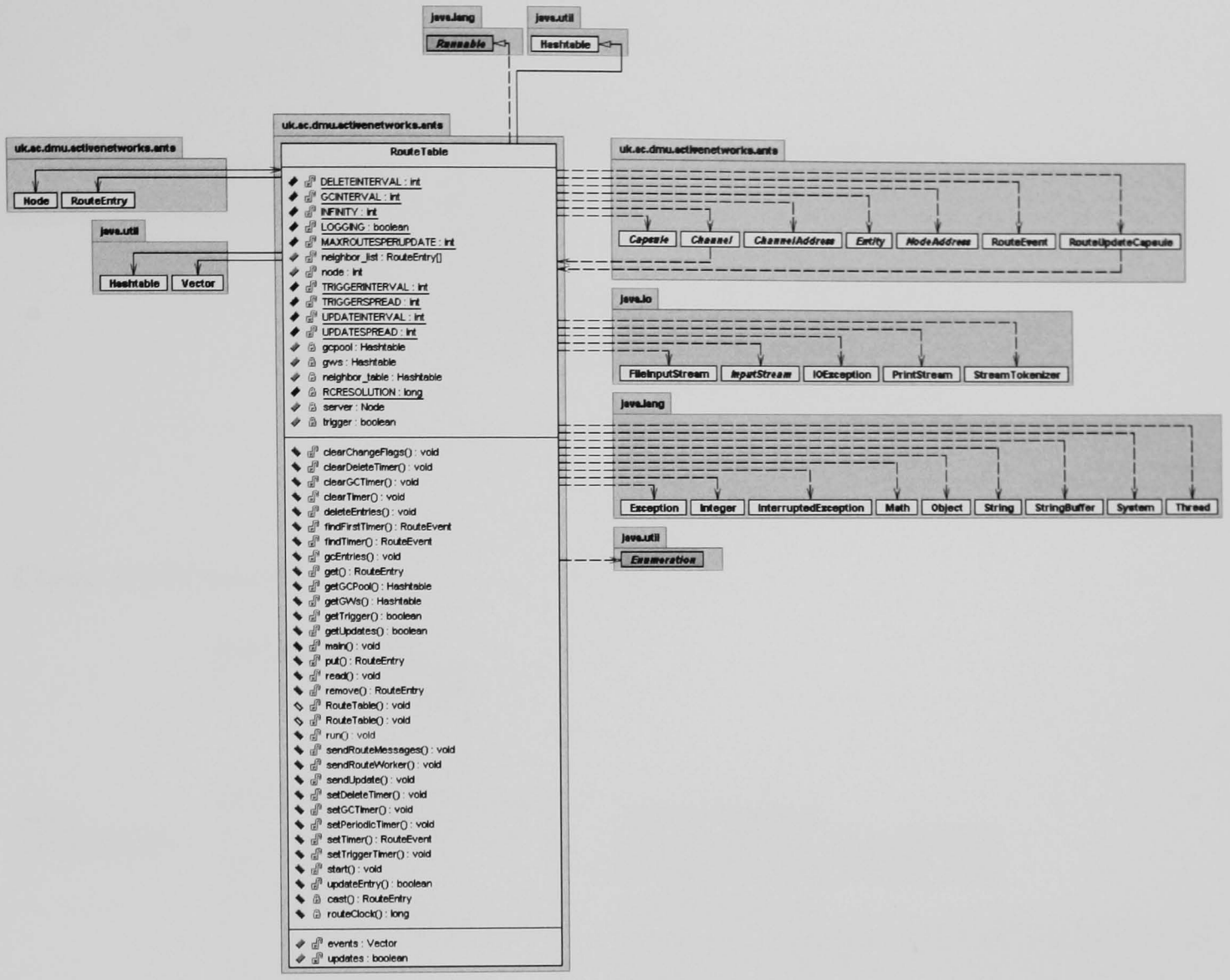
Class: RouteEvent



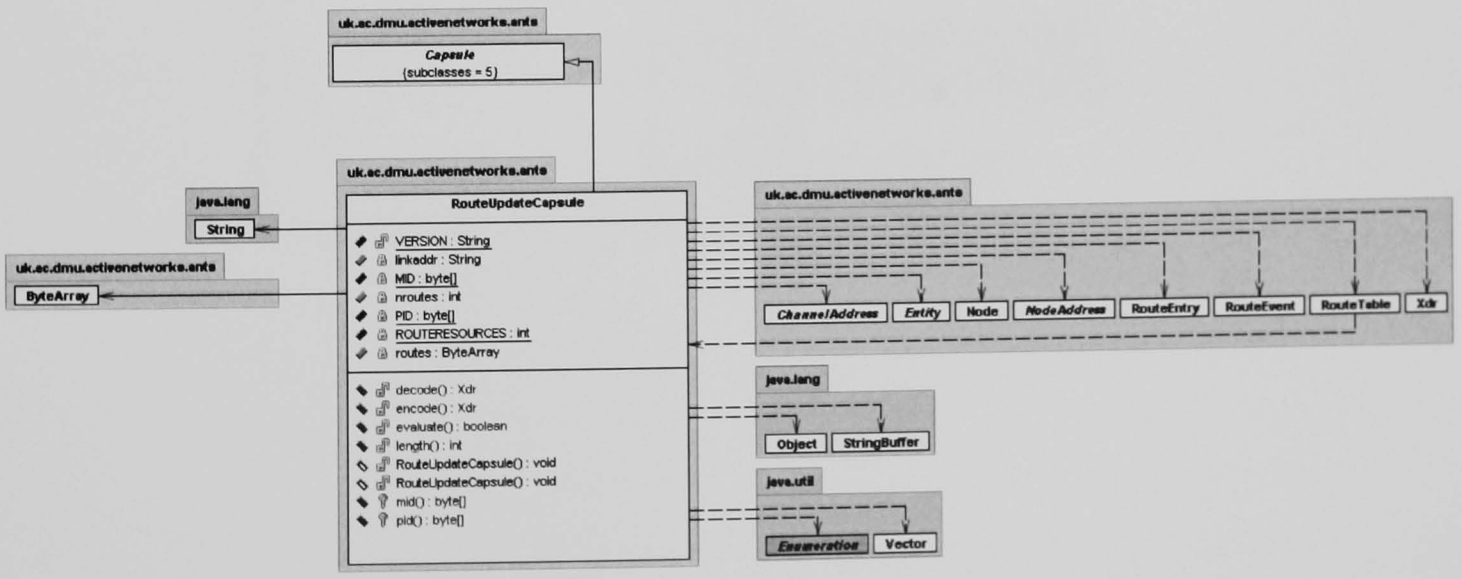
Class: RouteProtocol



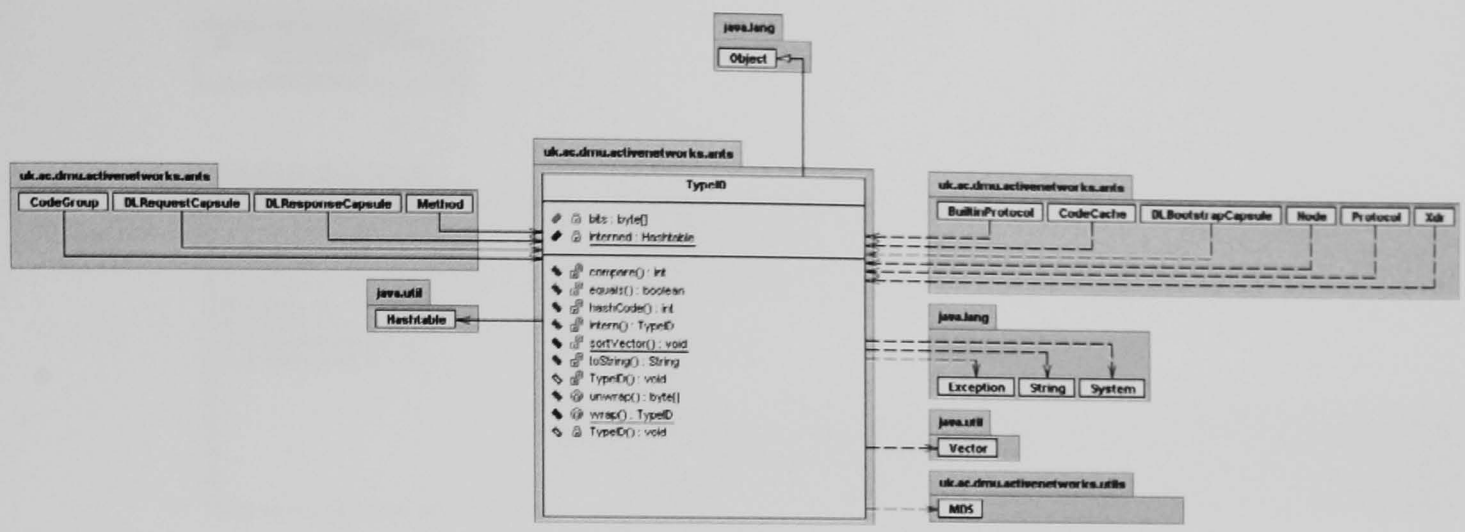
Class:RouteTable



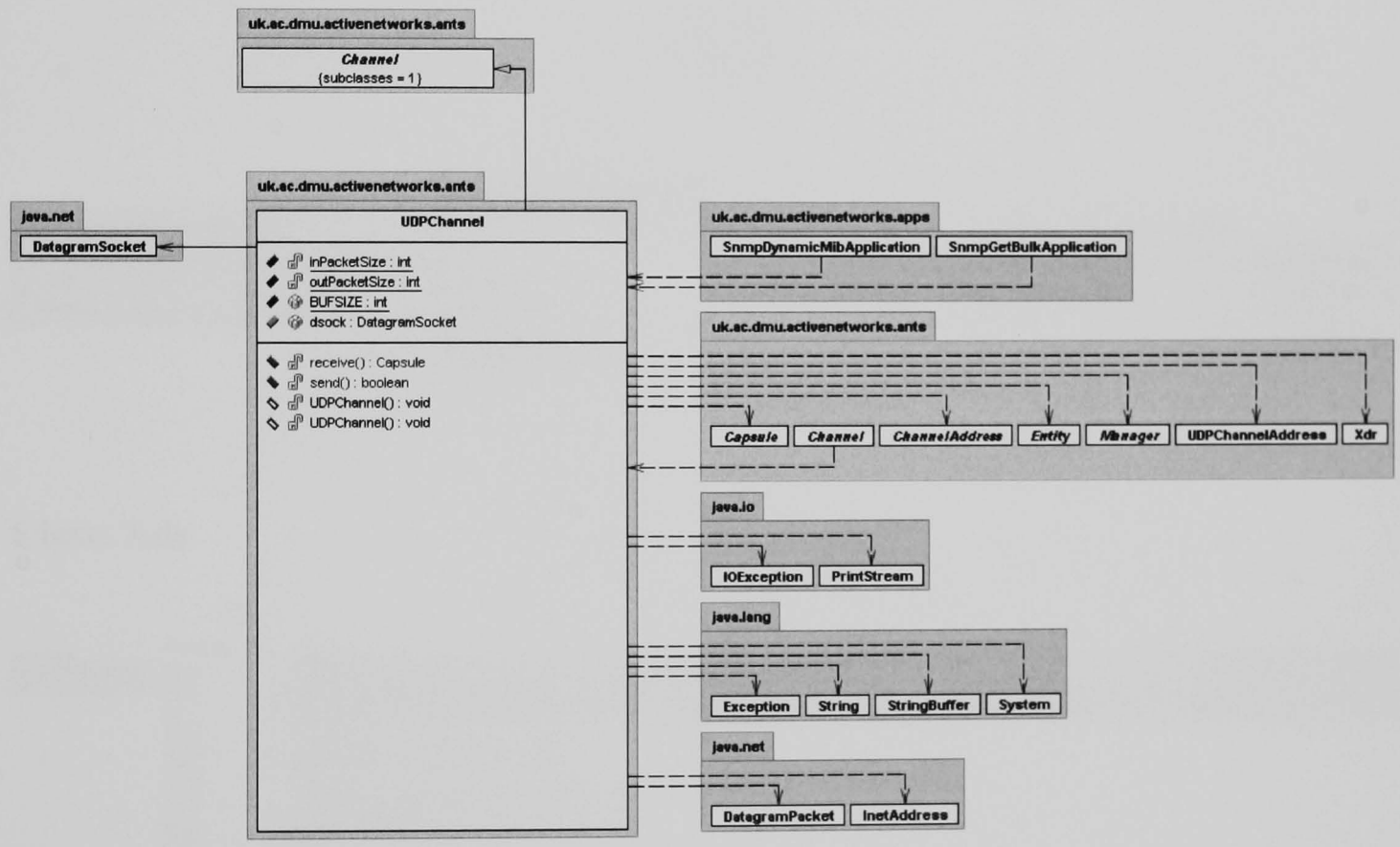
Class: RouteUpdateCapsule



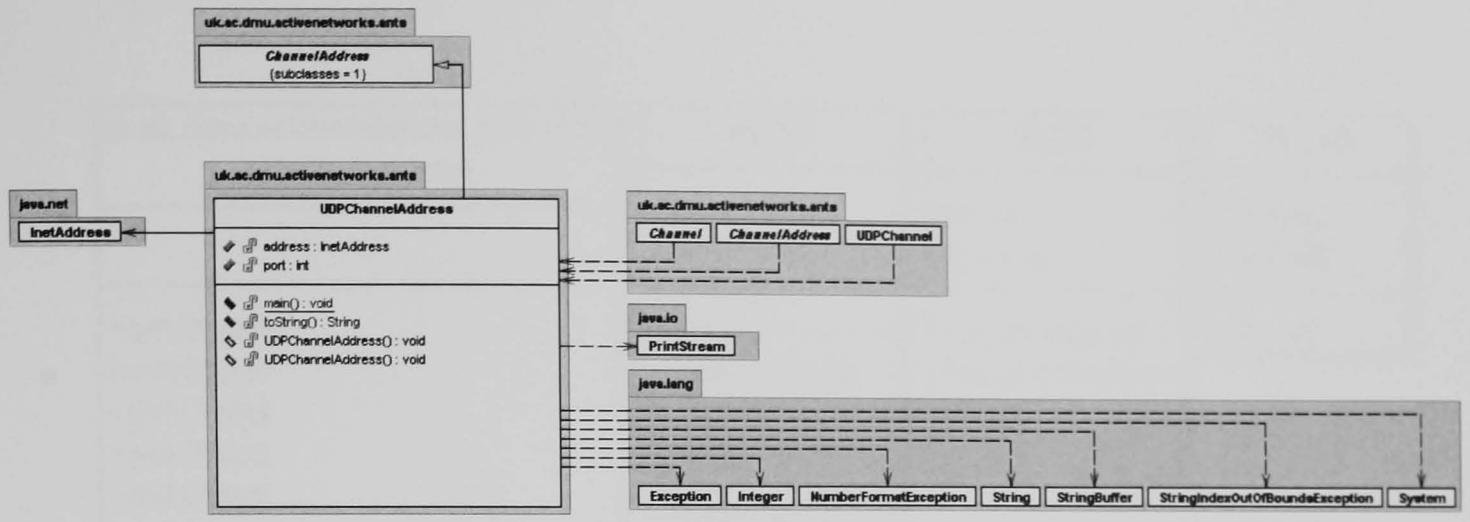
Class: TypeID



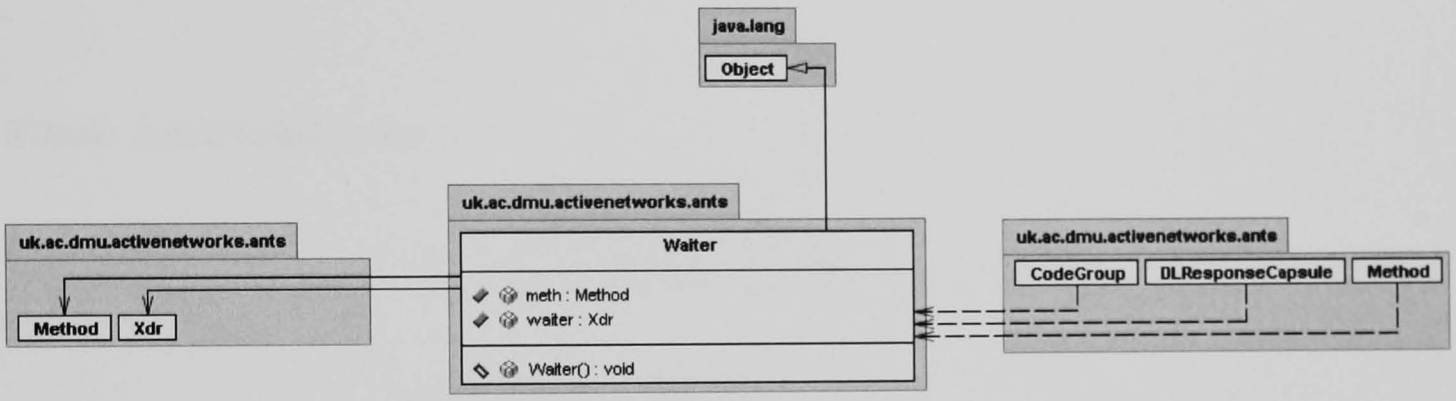
Class: UDPChannel



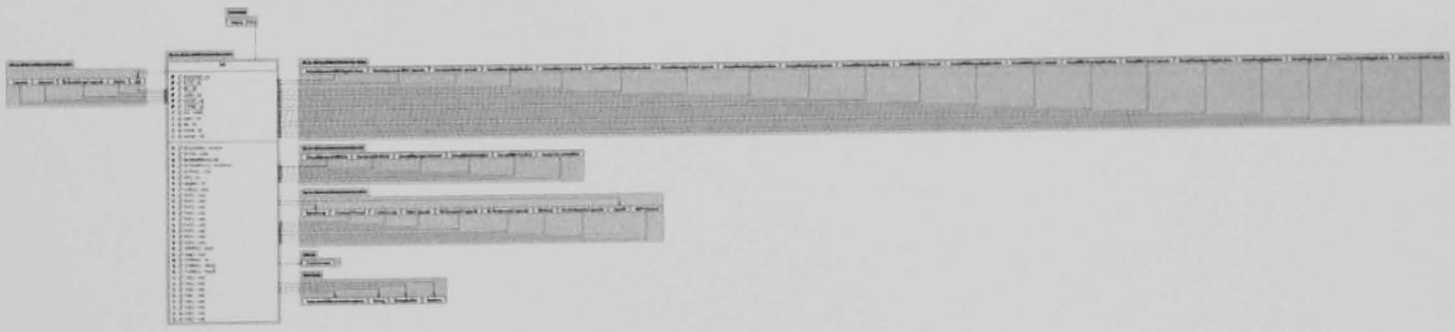
Class: UDPChannelAddress



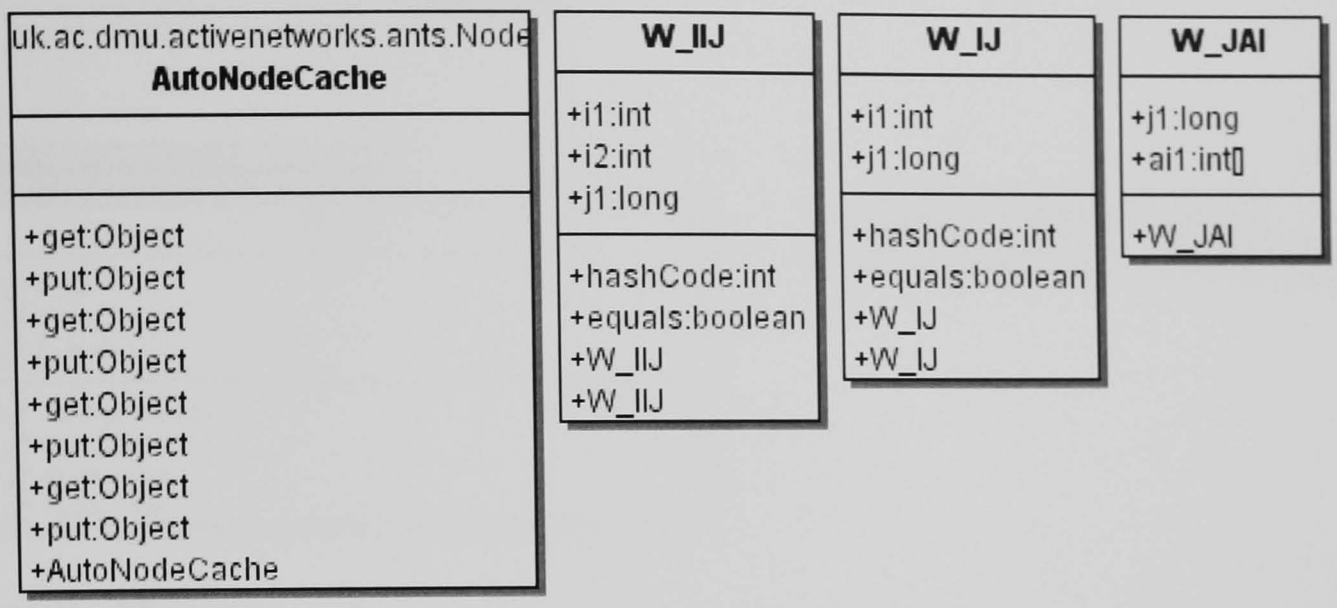
Class: Waiter



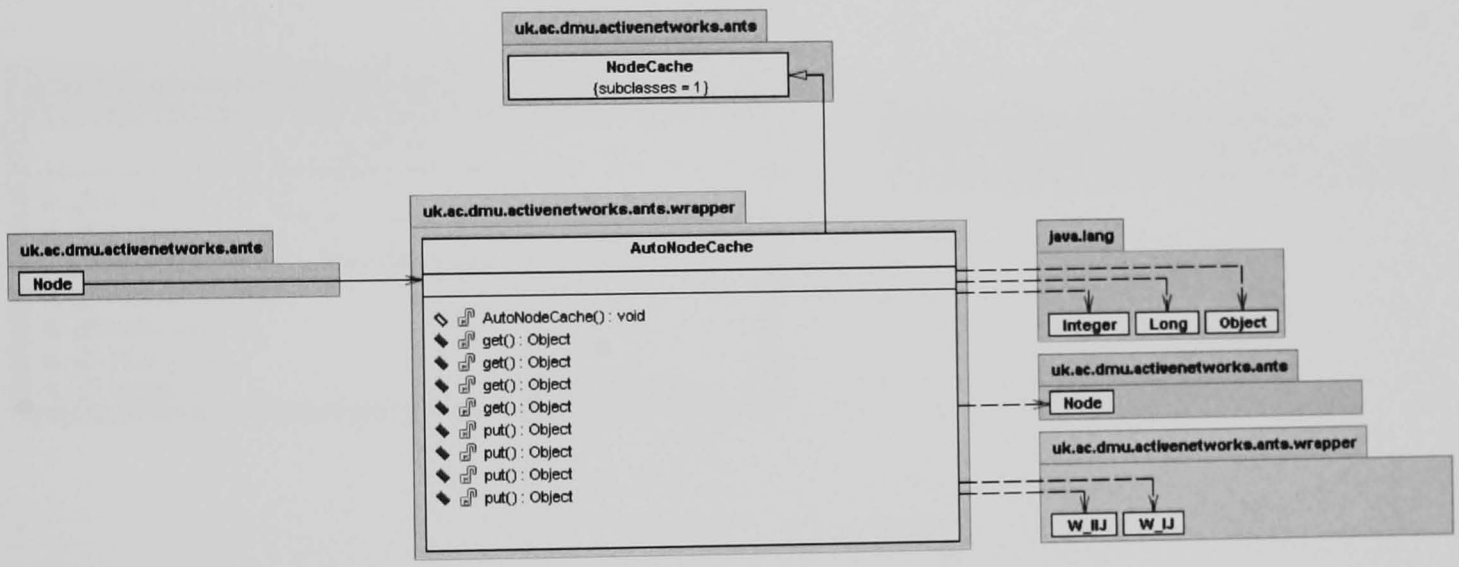
Class: Xdr



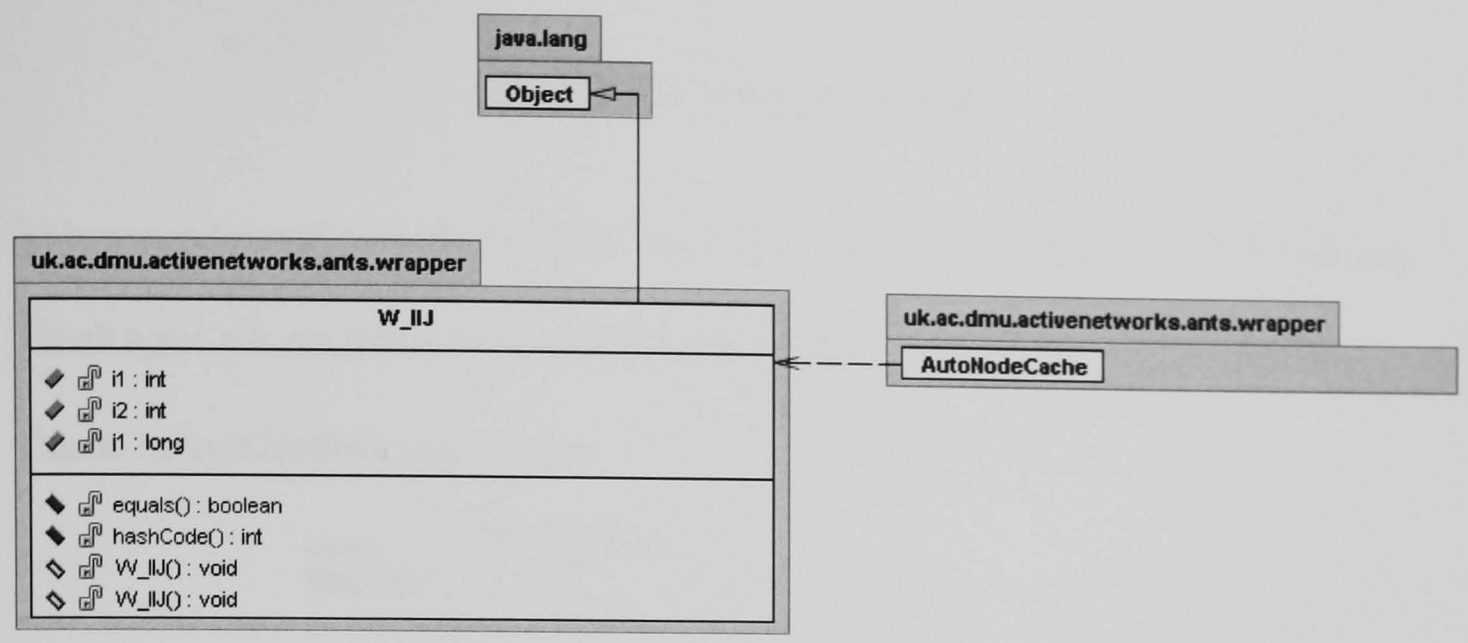
Package uk.ac.dmu.activenetworks.ants.wrapper



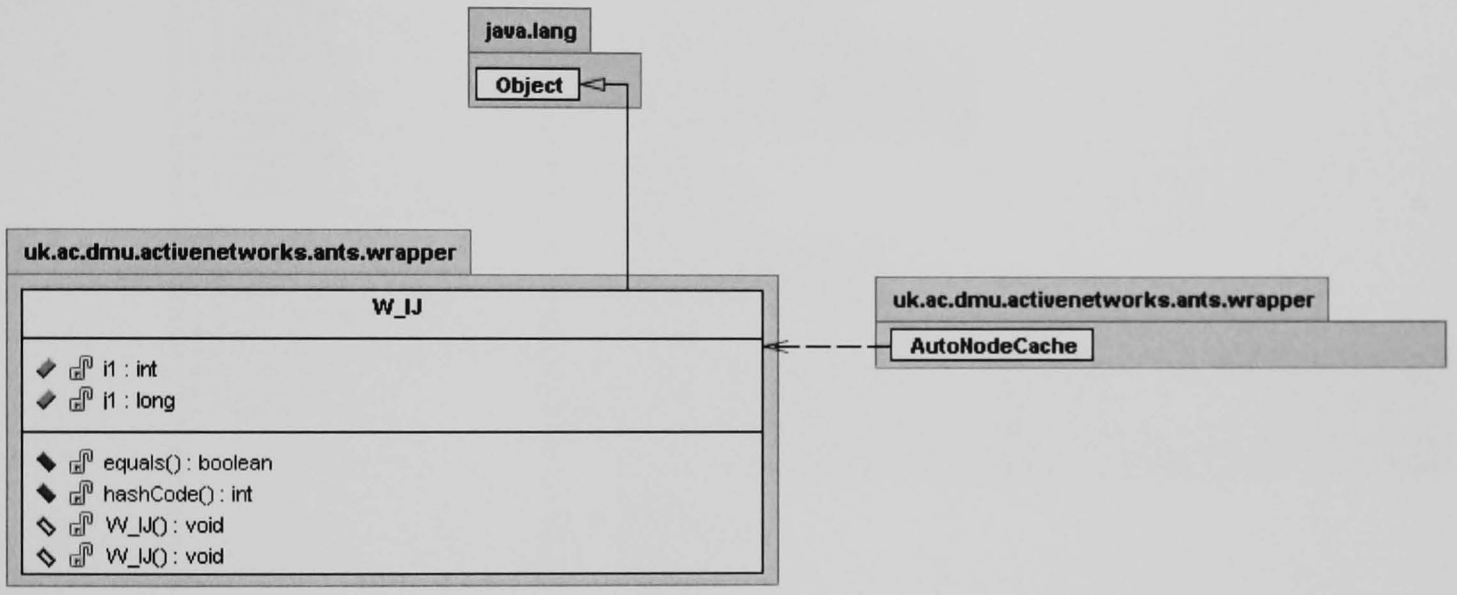
Class: AutoNodeCache



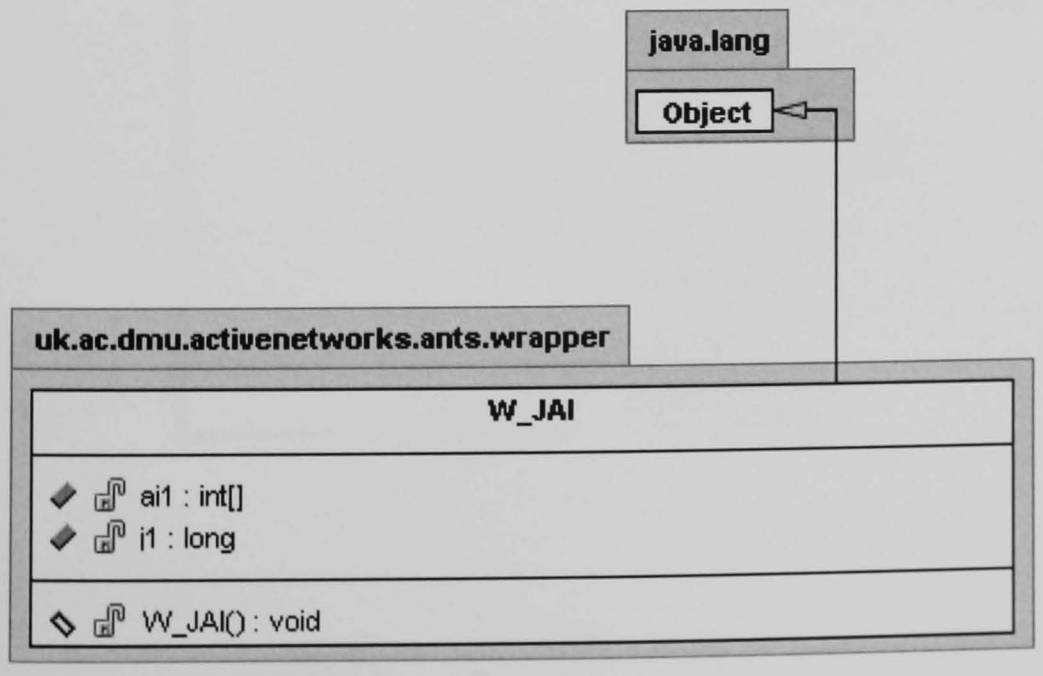
Class:W_IIJ



Class:W_IJ



Class: W_JAI

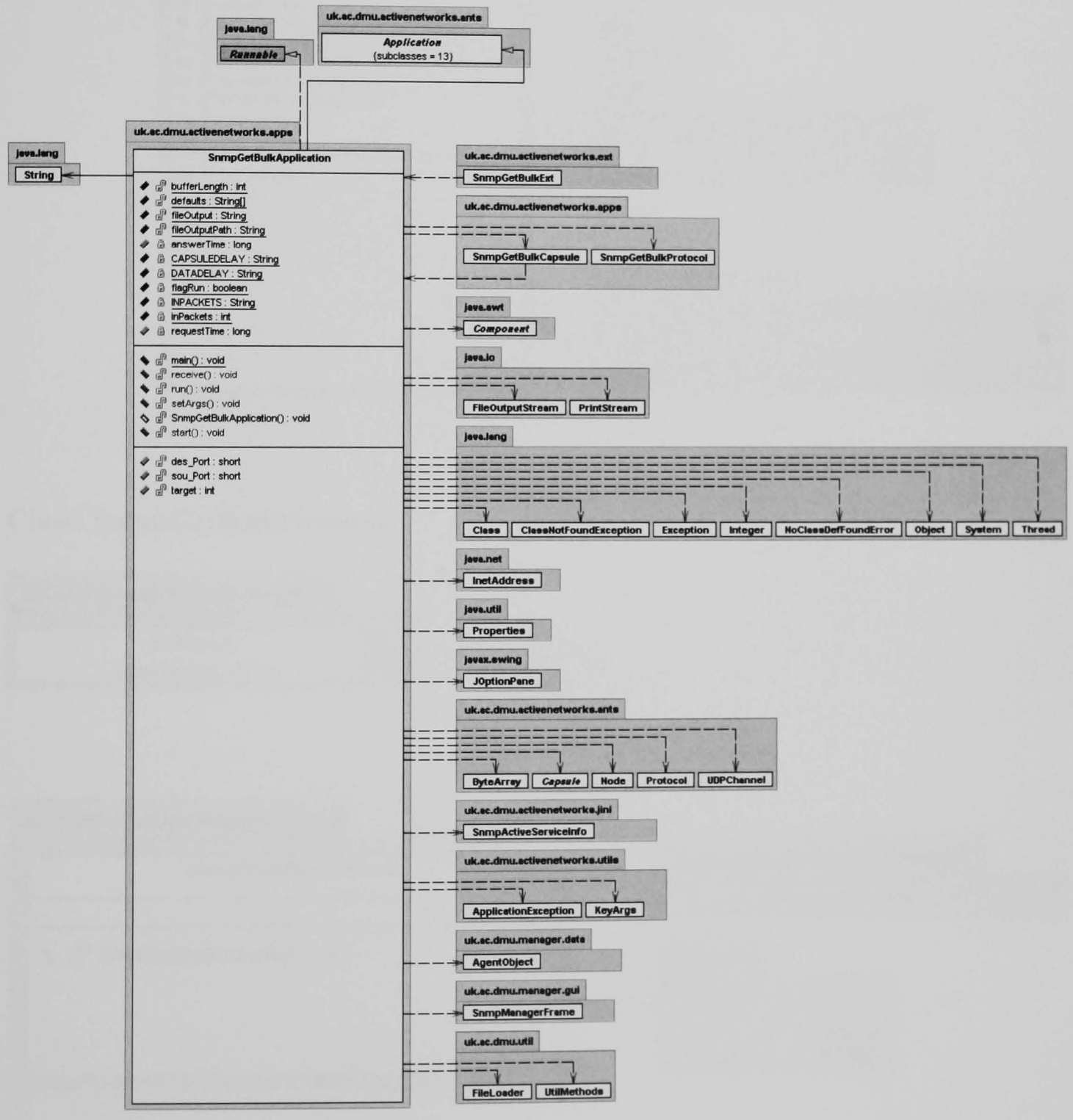


SNMP GETBULK SERVICE

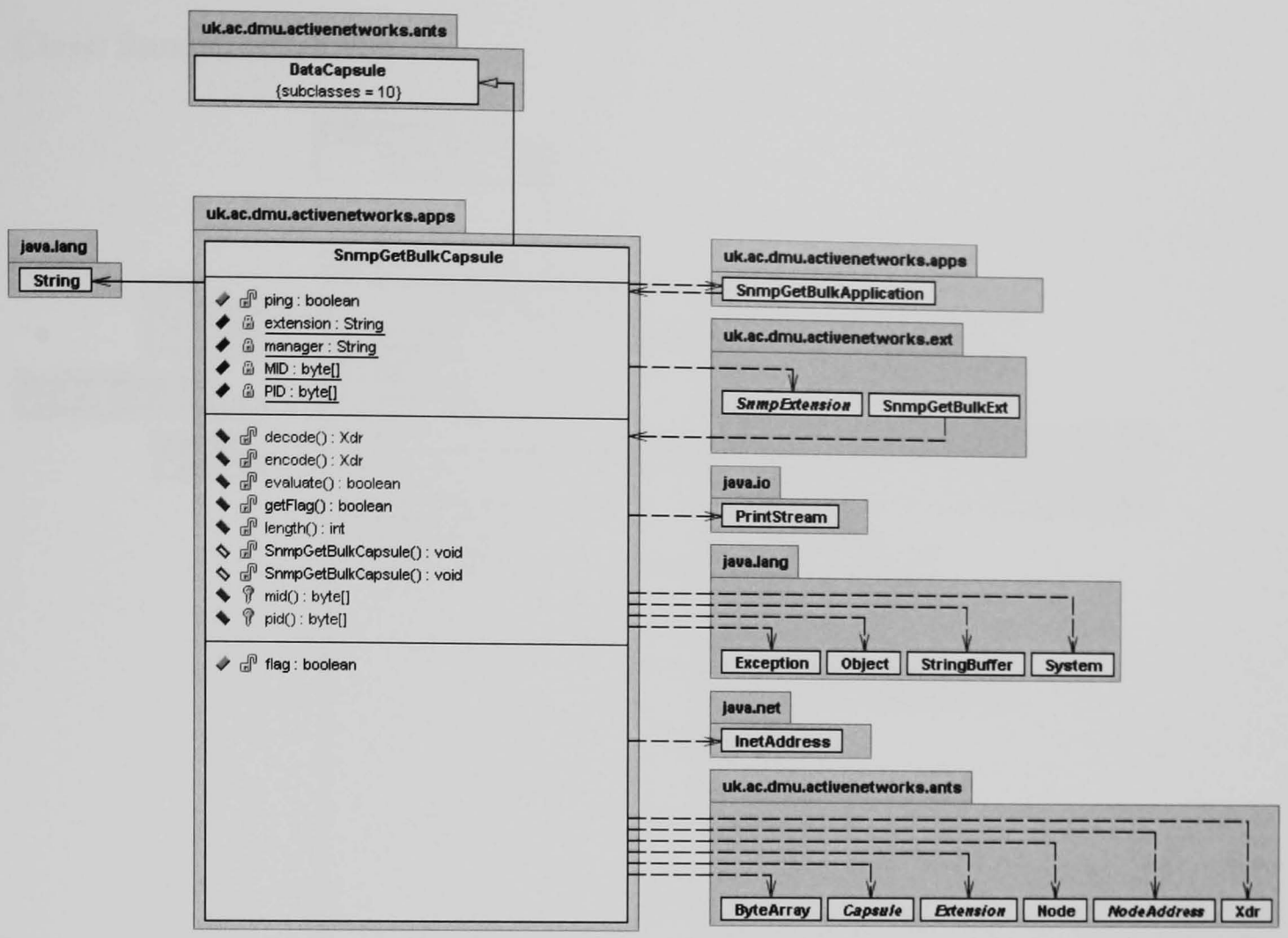
This appendix will provide the UML diagrams related to the SNMP GetBulk Service.

Package: uk.ac.dmu.activenetworks.apps

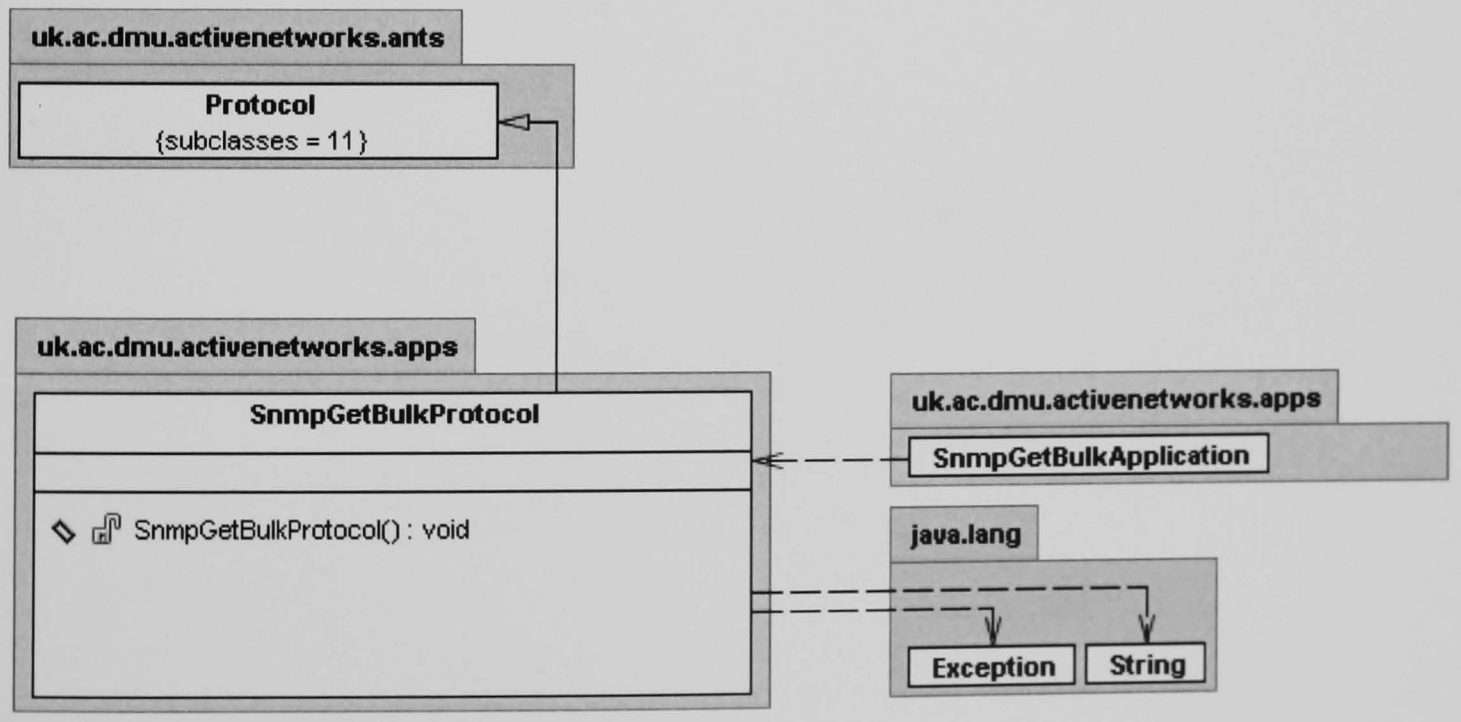
Class: SnmpGetBulkApplication



Class: SnmpGetBulkCapsule

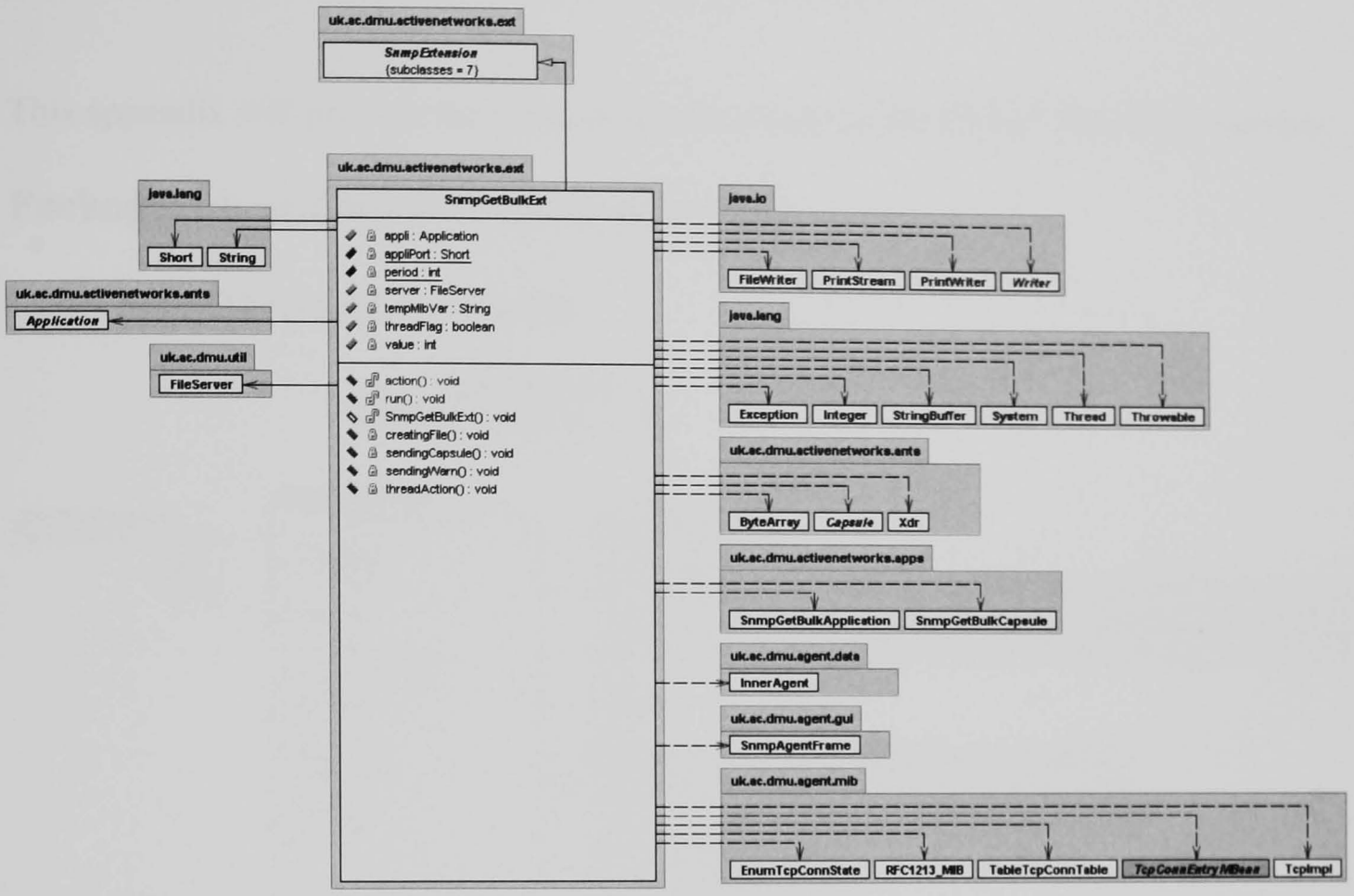


Class: SnmpGetBulkProtocol



package: uk.ac.dmu.apps.ext

Class: SnmpGetBulkExt

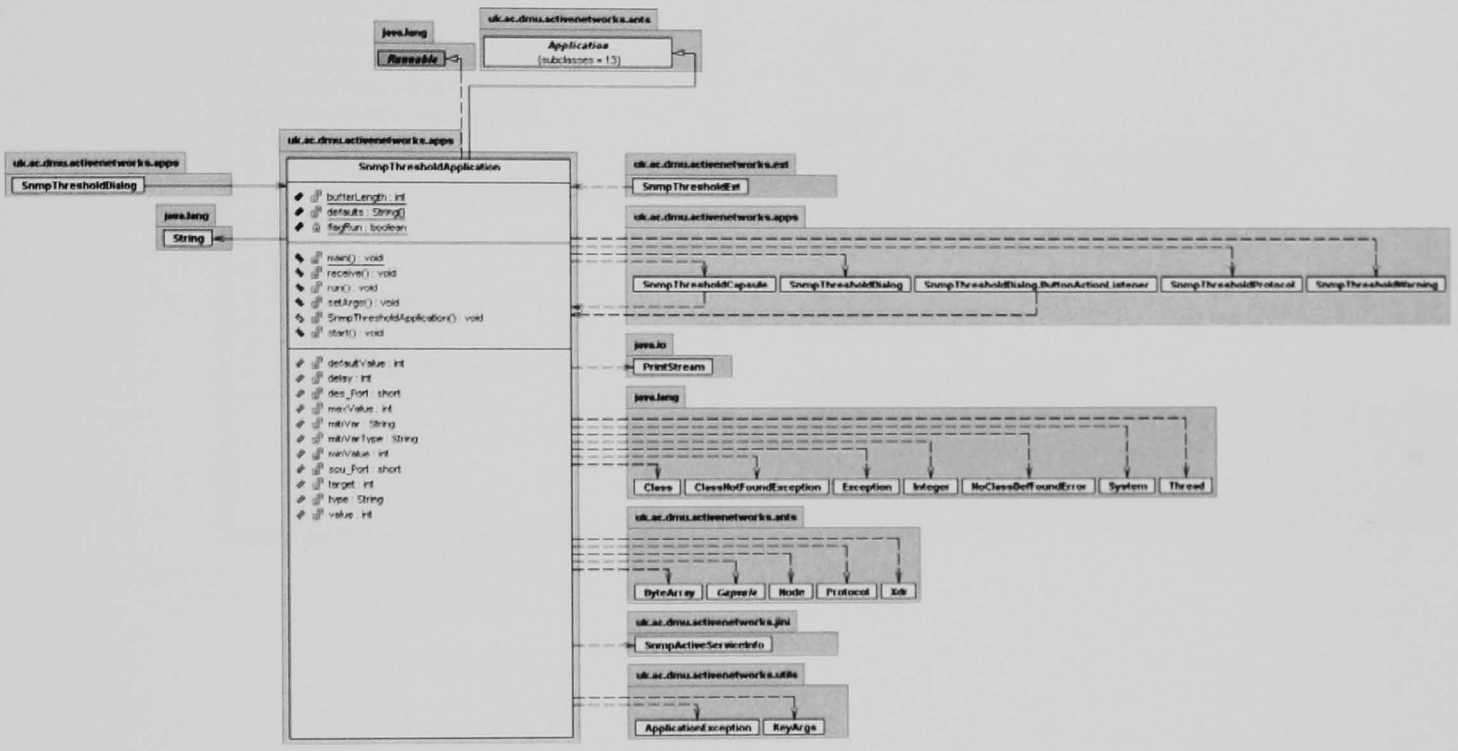


SNMP THRESHOLD SERVICE

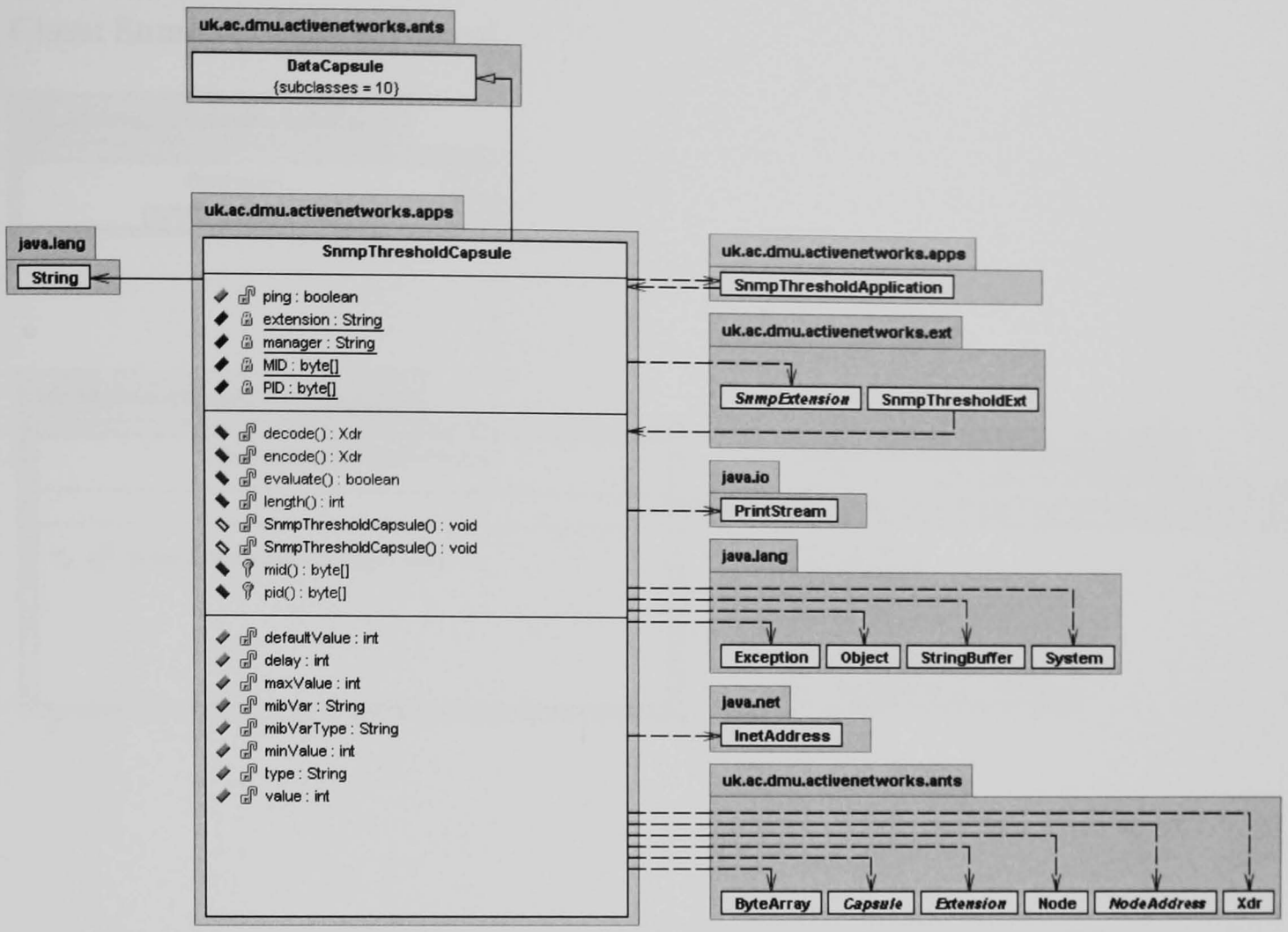
This appendix will provide the UML diagrams related to the SNMP Threshold Service.

Package: uk.ac.dmu.activenetworks.apps

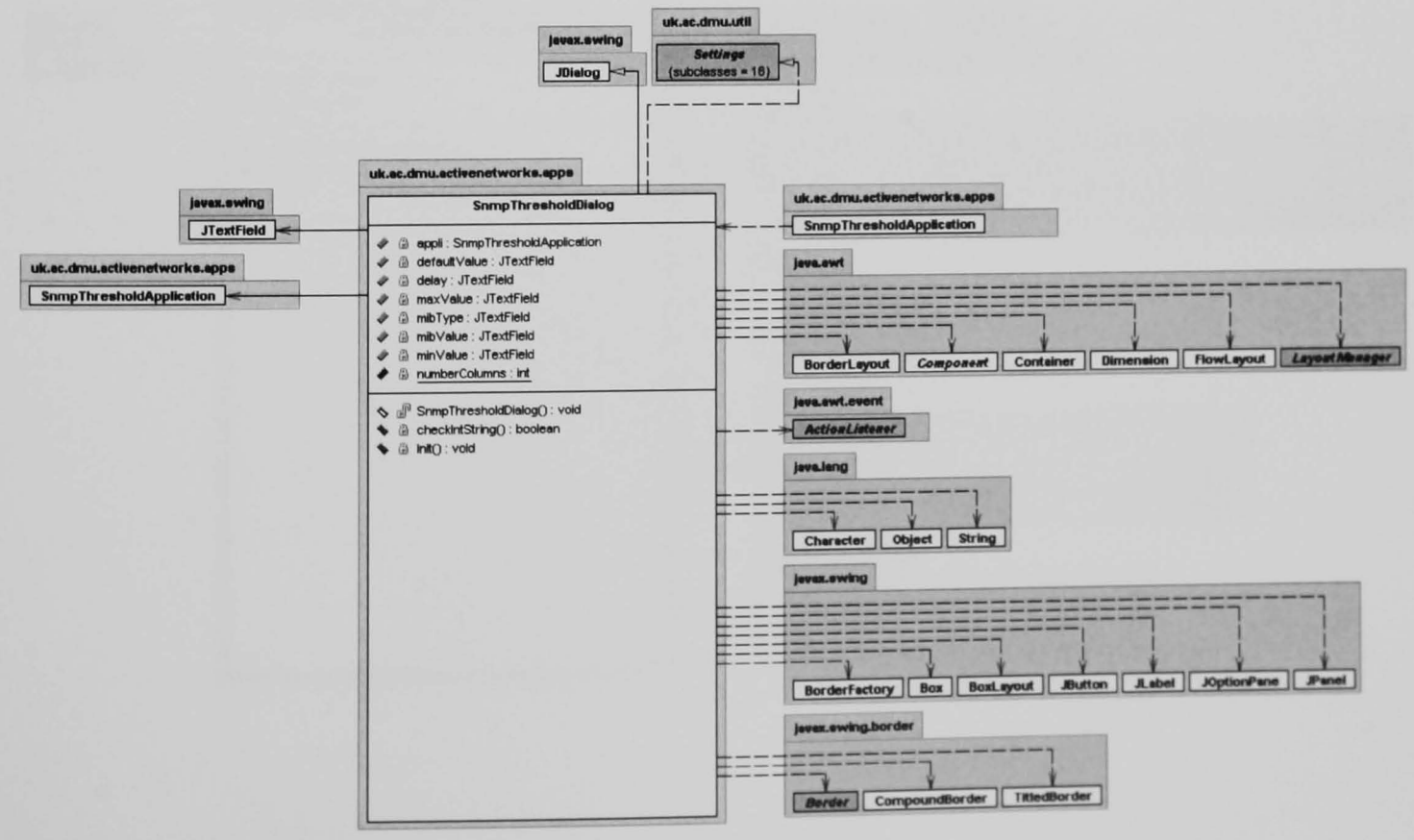
Class: SnmpThresholdApplication



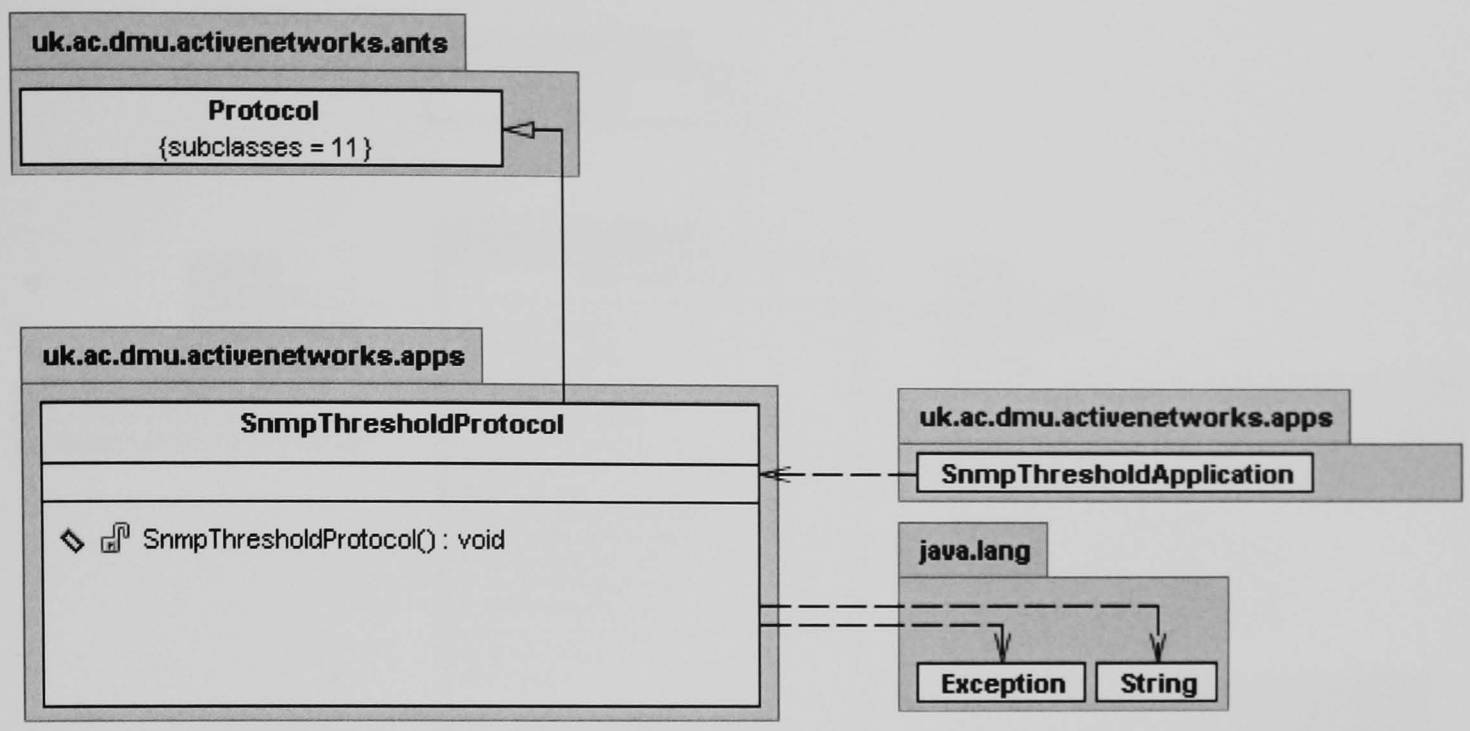
Class: SnmpThresholdCapsule



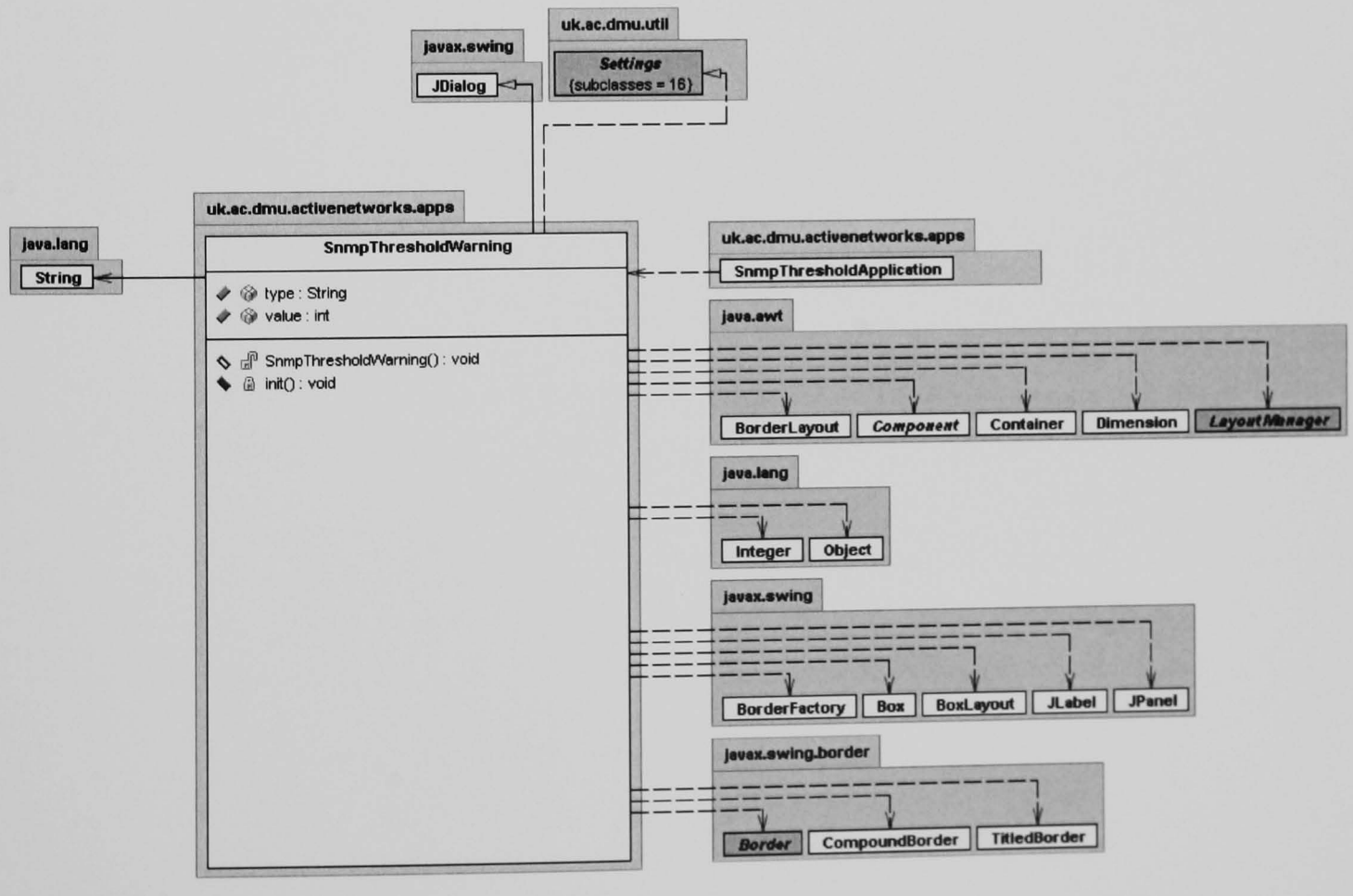
Class: SnmpThresholdDialog



Class: SnmpThresholdProtocol

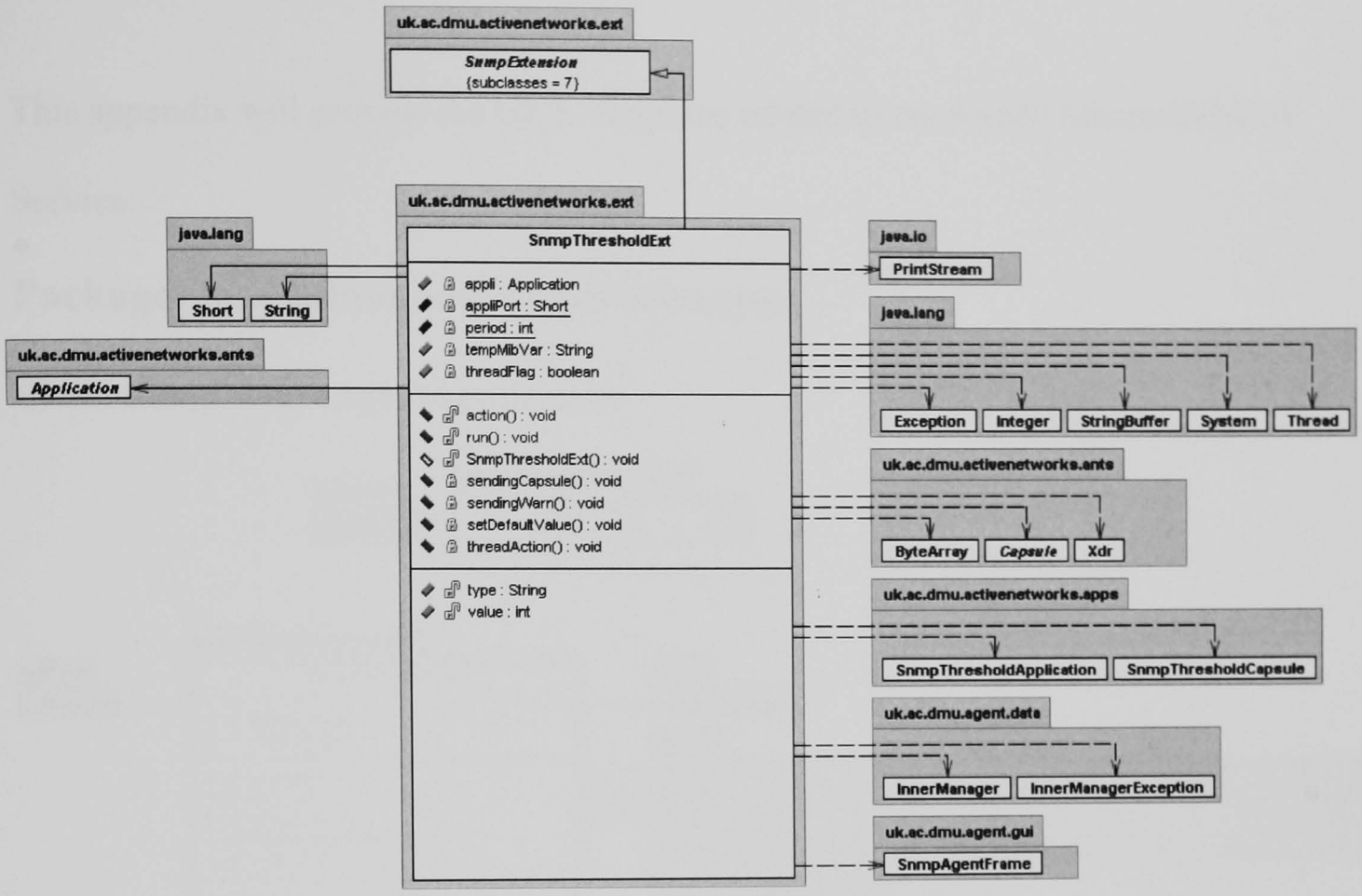


Class: SnmpThresholdWarning



package: uk.ac.dmu.apps.ext

Class: SnmpThresholdExt

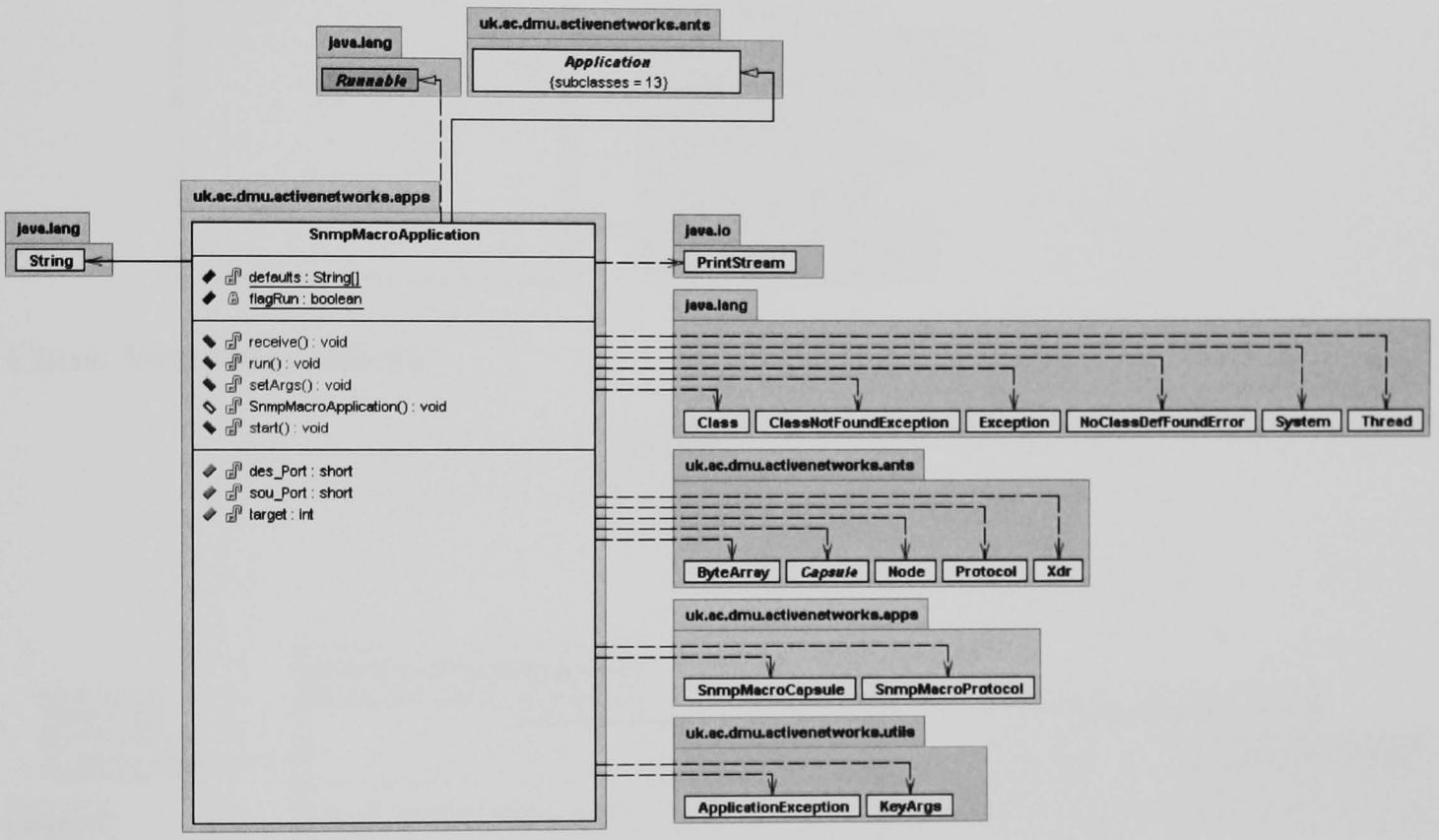


SNMP MACRO NETWORK SERVICE

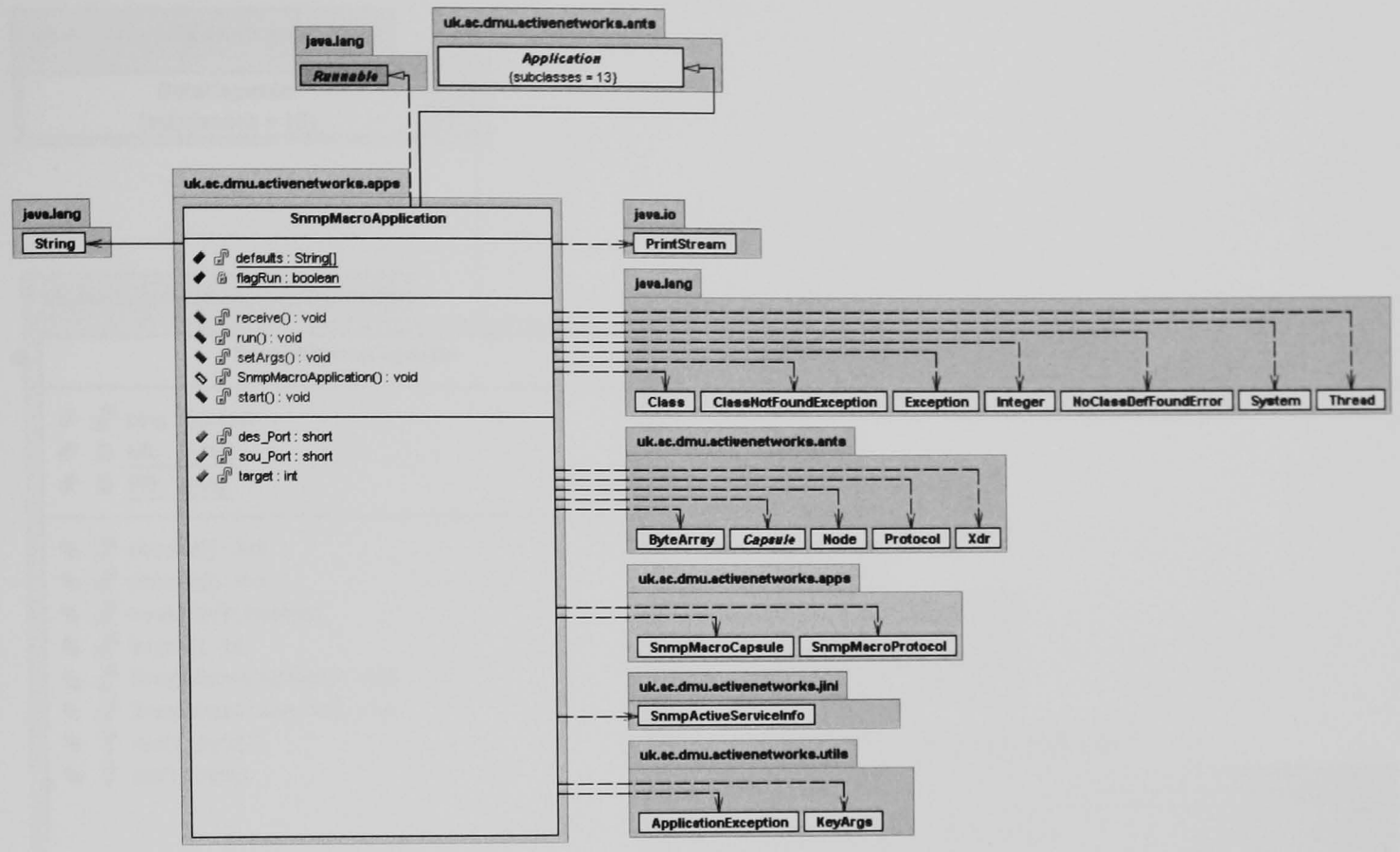
This appendix will provide the UML diagrams related to the SNMP Macro Network Service.

Package: uk.ac.dmu.activenetworks.apps

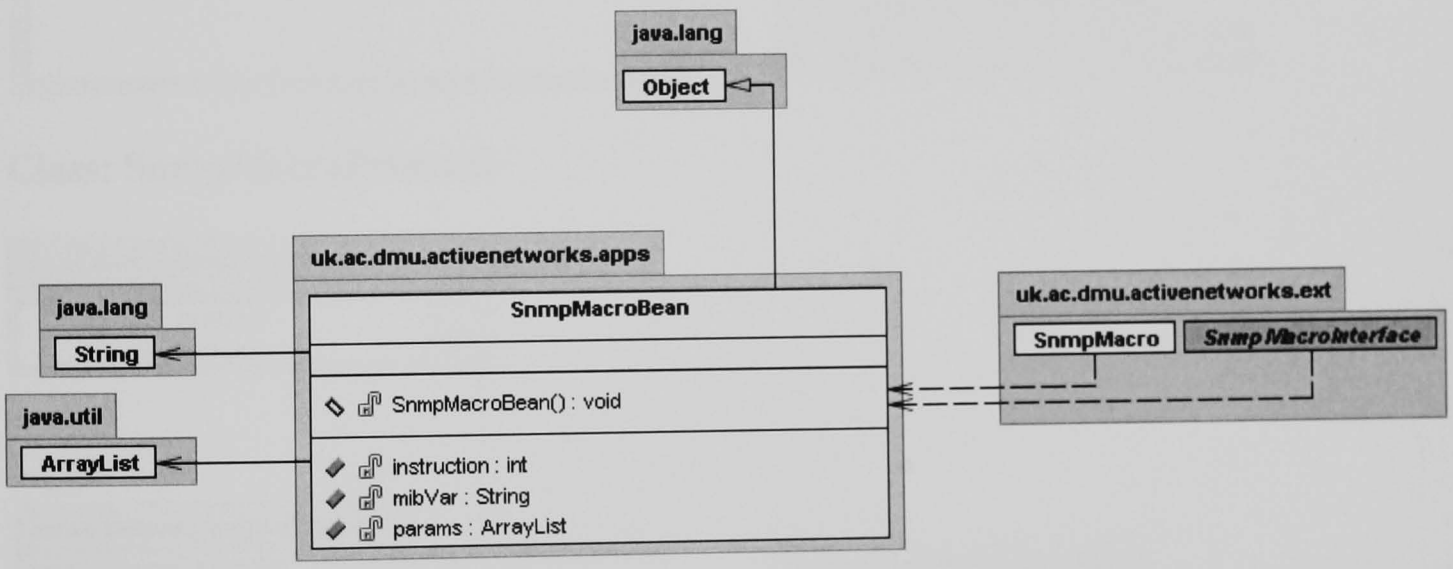
Class: SnmpMacroAgentApplication



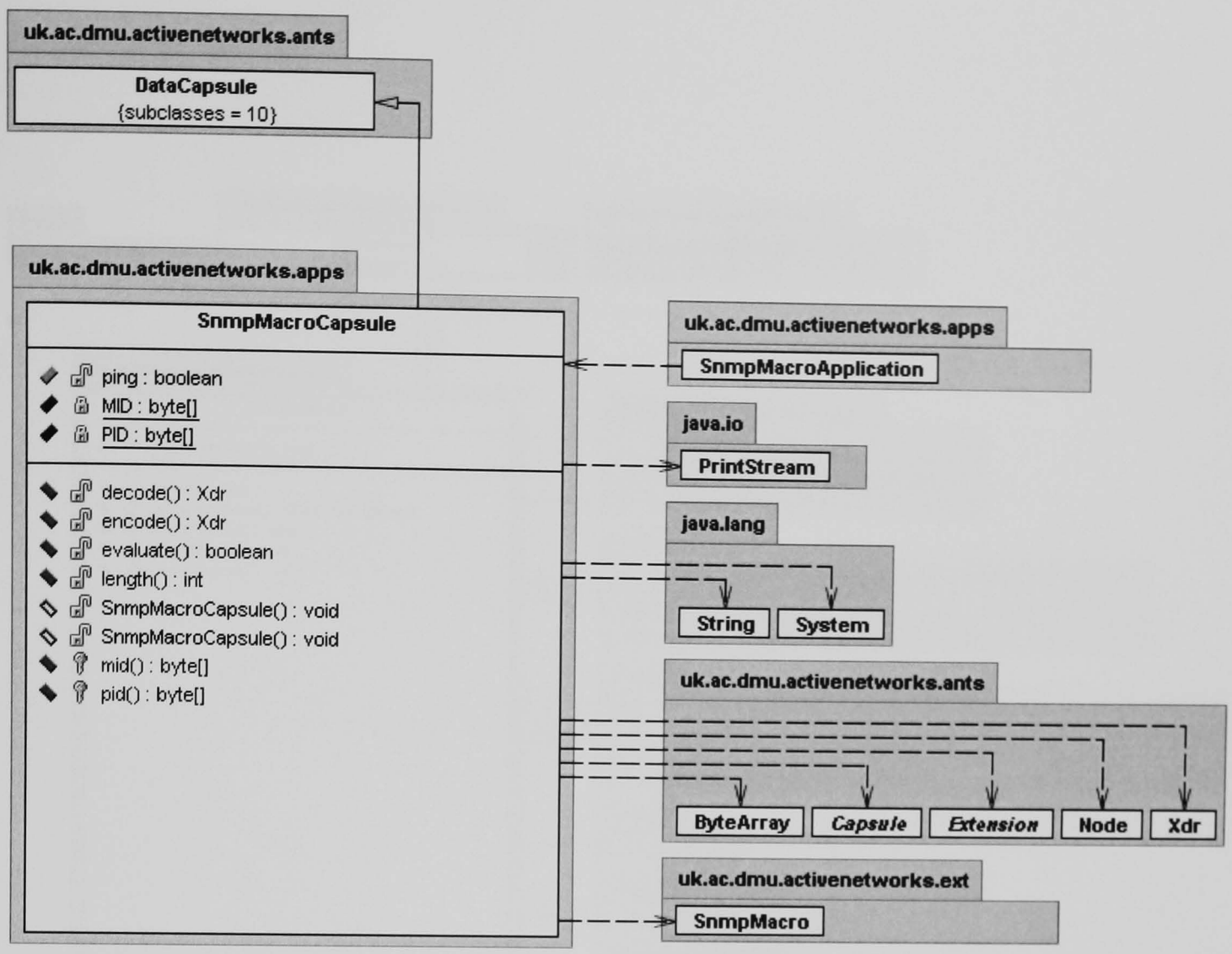
Class: SnmpMacroApplication



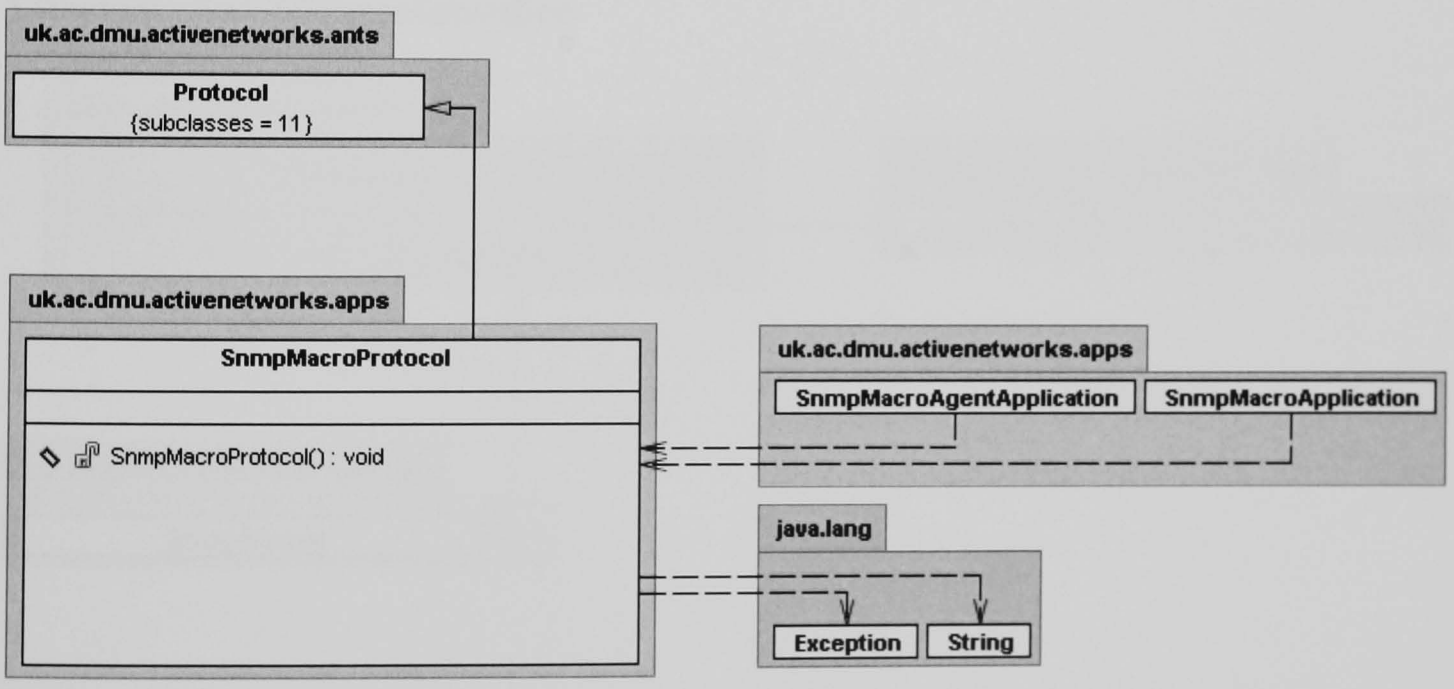
Class: SnmpMacroBean



Class: SnmpMacroCapsule

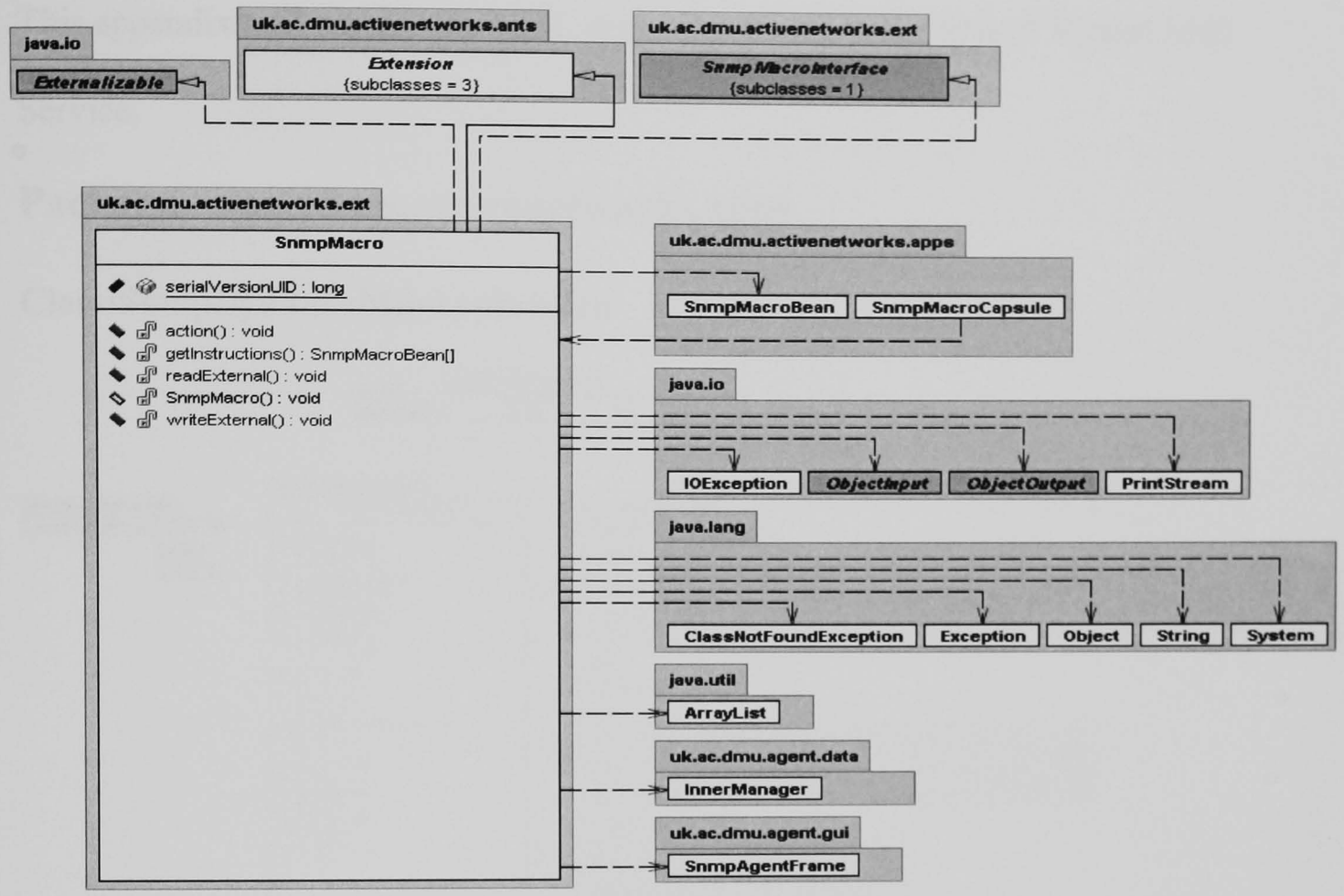


Class: SnmpMacroProtocol

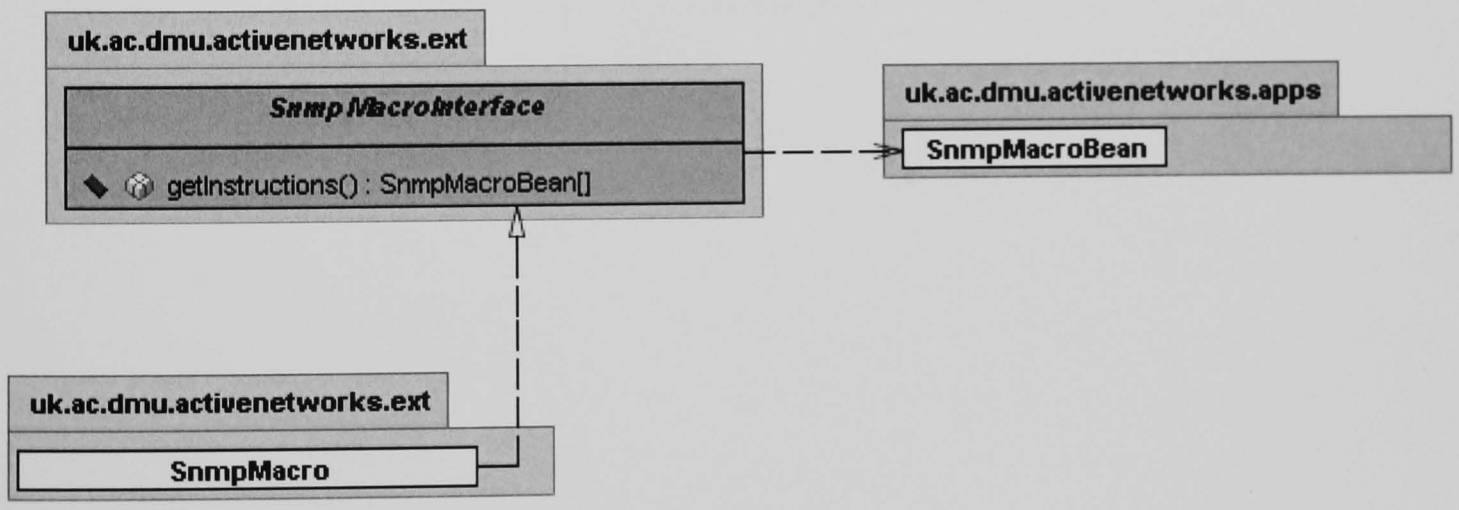


package: uk.ac.dmu.apps.ext

Class: SnmpMacro



Interface: SnmpMacroInterface

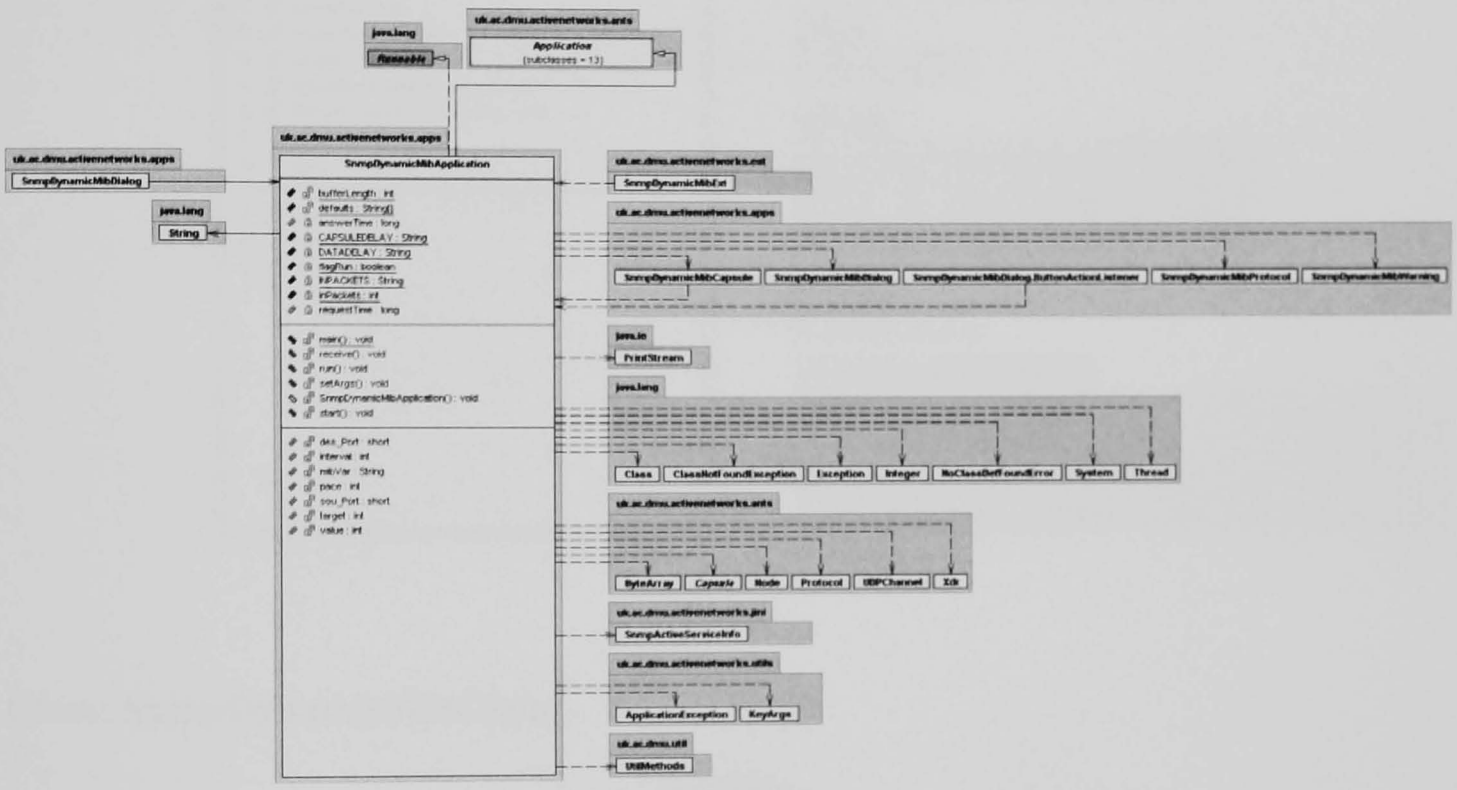


SNMP VIRTUAL MIB SERVICE

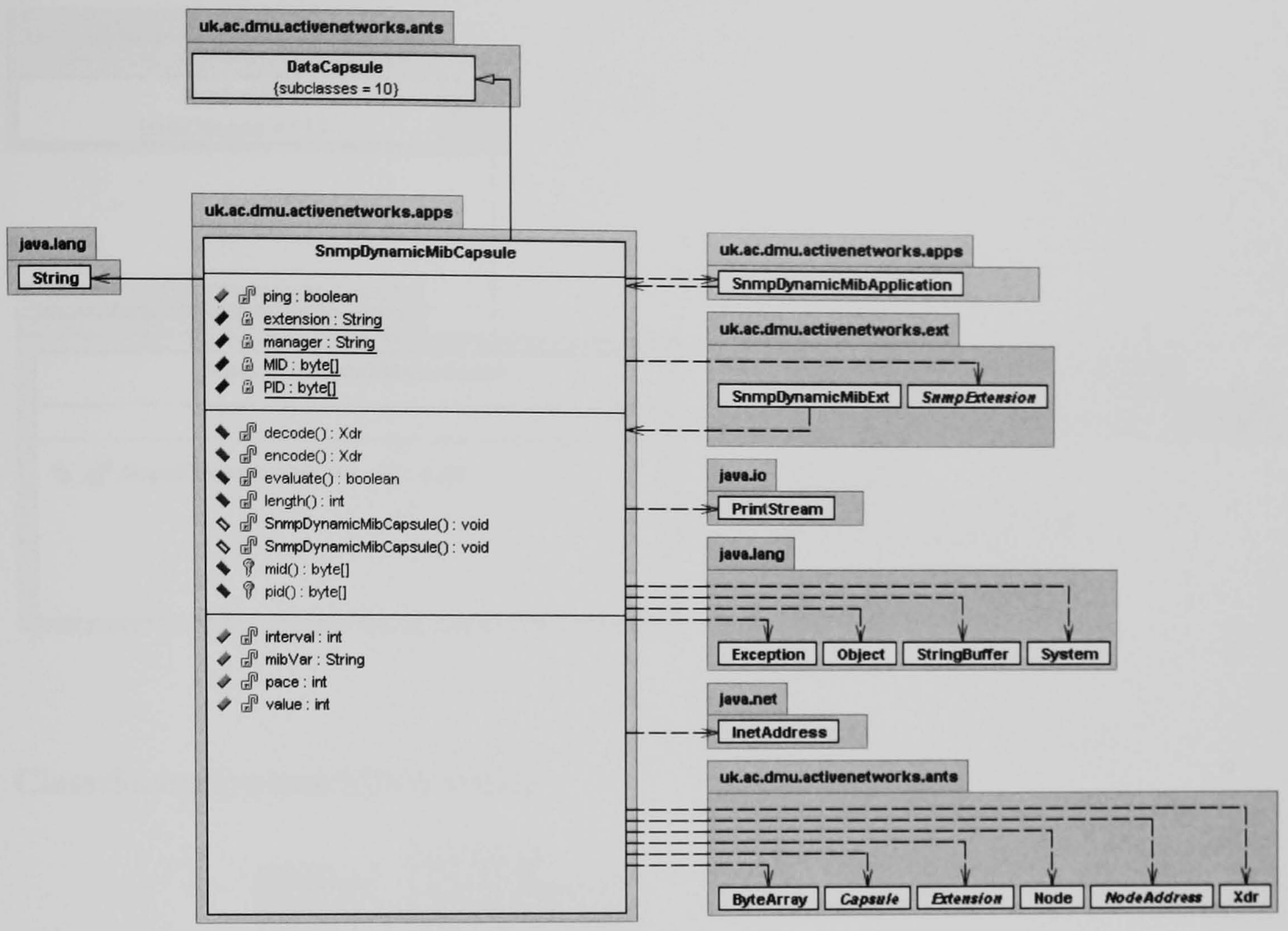
This appendix will provide the UML diagrams related to the SNMP Virtual MIB Service.

Package: uk.ac.dmu.activenetworks.apps

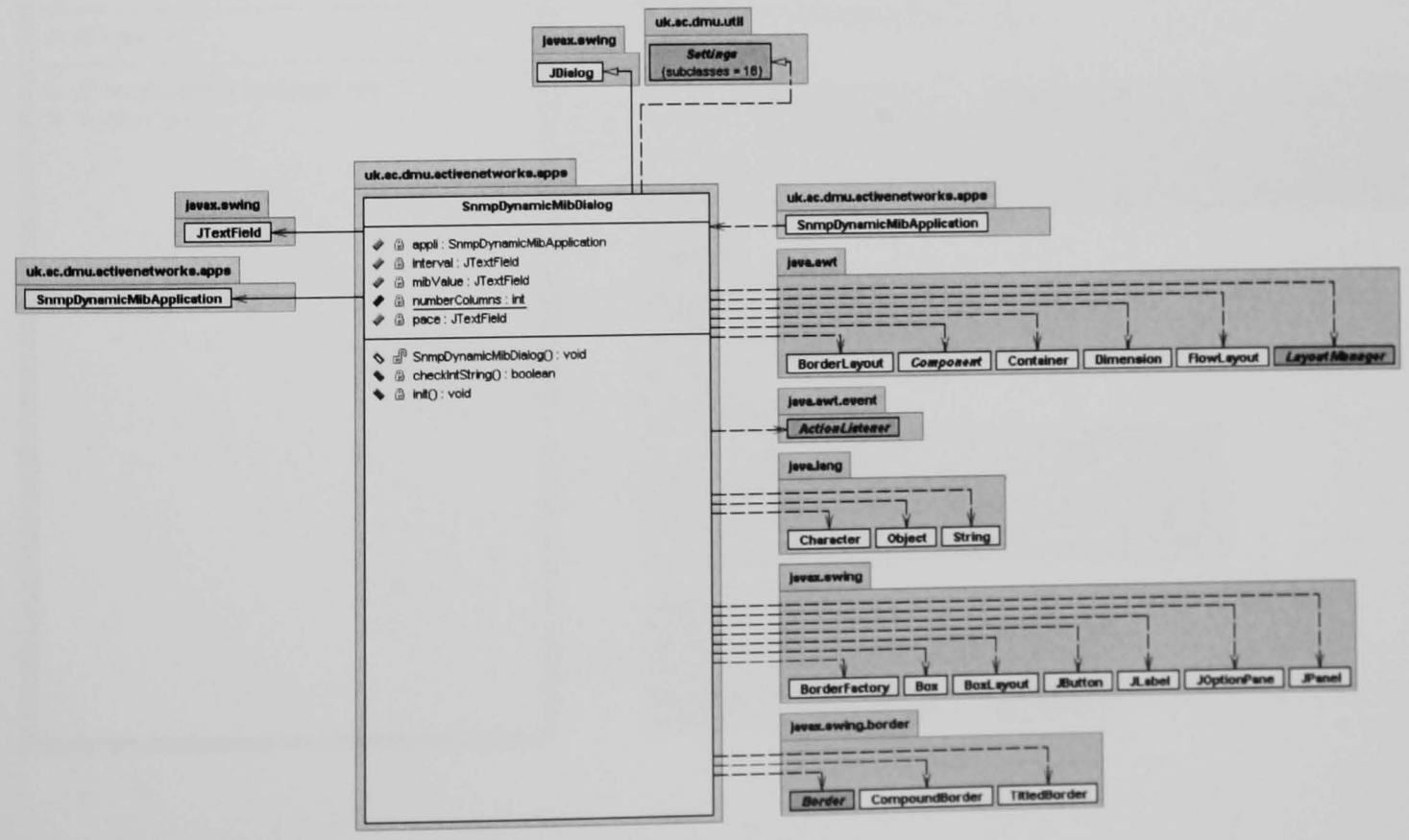
Class:SnmpDynamicMibApplication



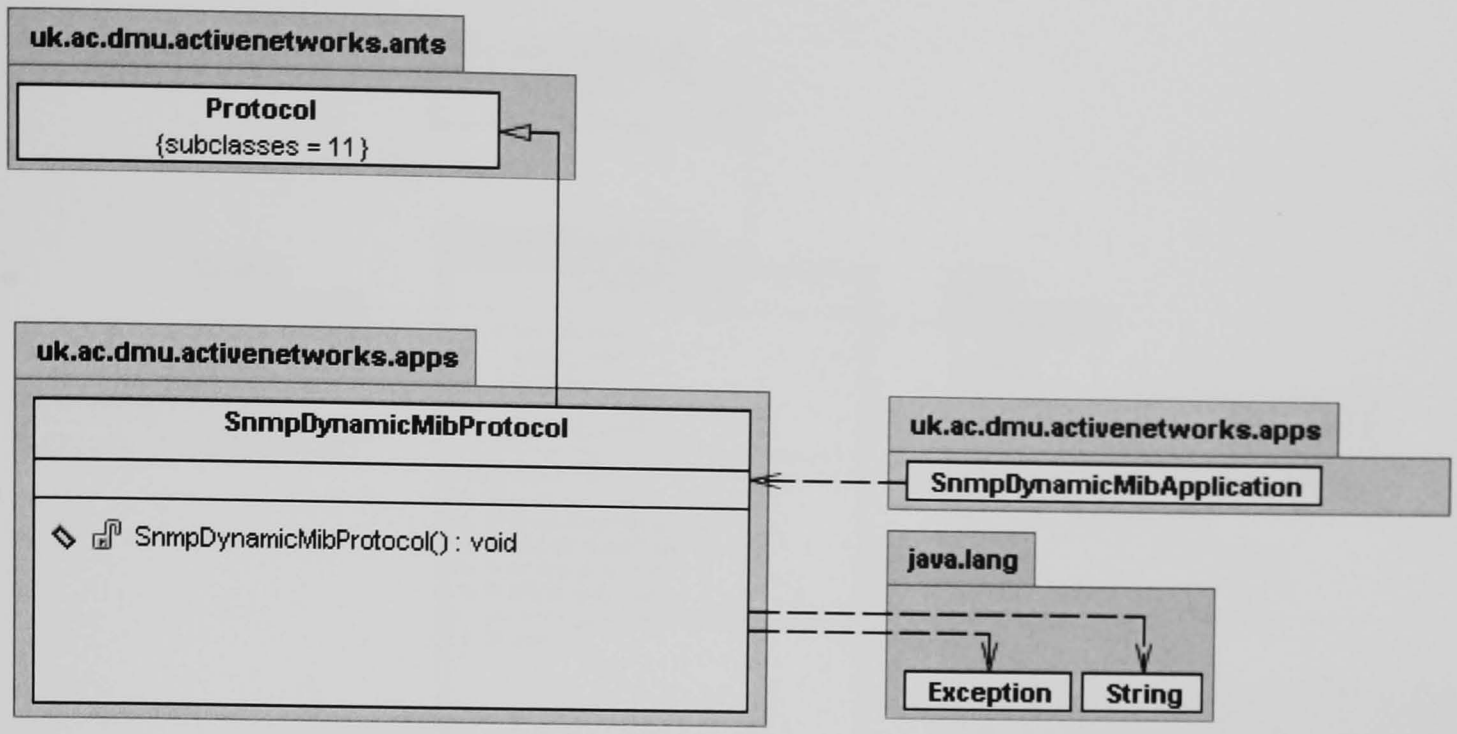
Class: SnmpDynamicMibCapsule



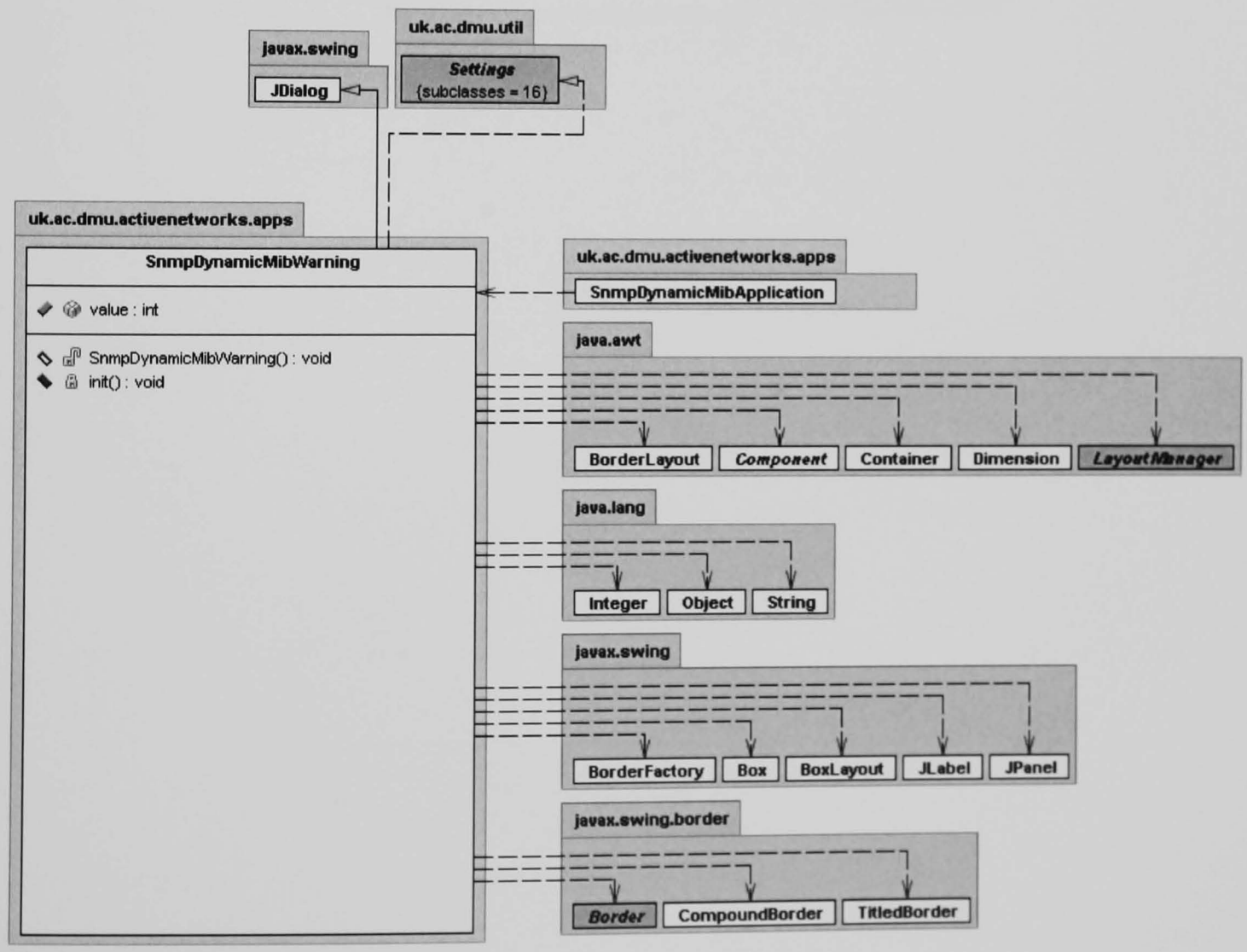
Class: SnmpDynamicMibDialog



Class: SnmpDynamicMibProtocol

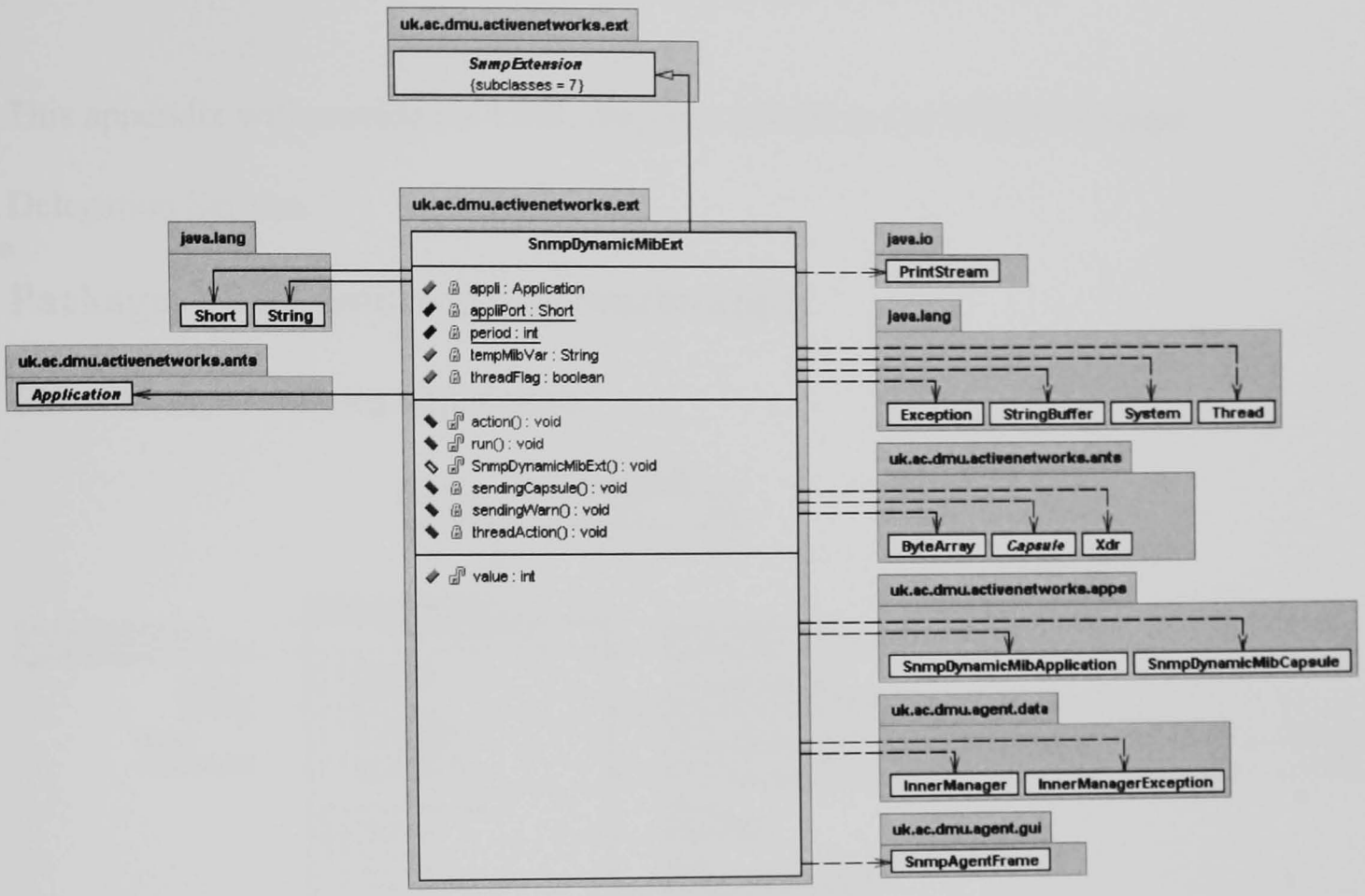


Class:SnmpDynamicMibWarning



package: uk.ac.dmu.apps.ext

Class: SnmpDynamicMibExt

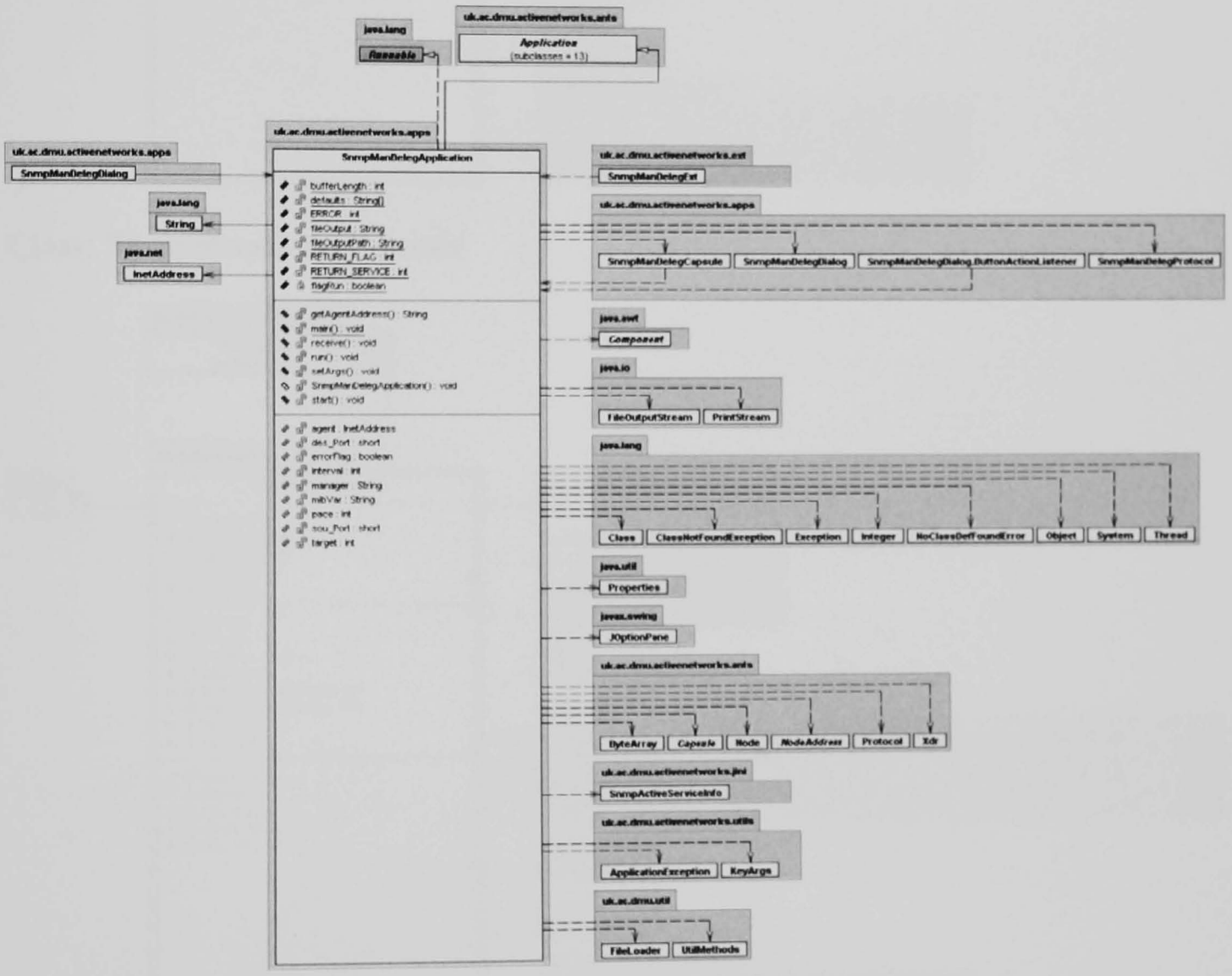


SNMP MANAGER DELEGATION SERVICE

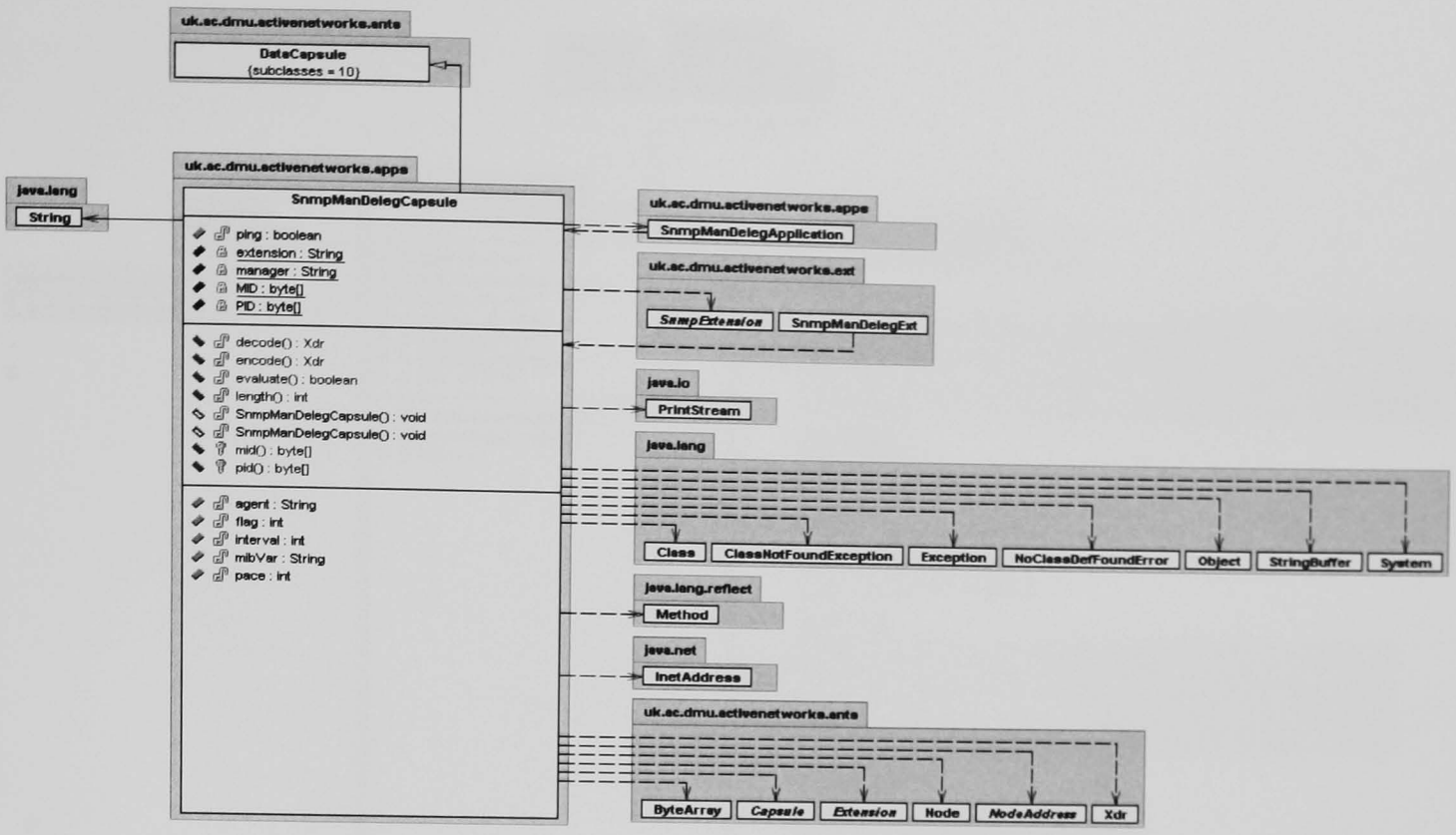
This appendix will provide the UML diagrams related to the SNMP Manager Delegation Service.

Package: uk.ac.dmu.activenetworks.apps

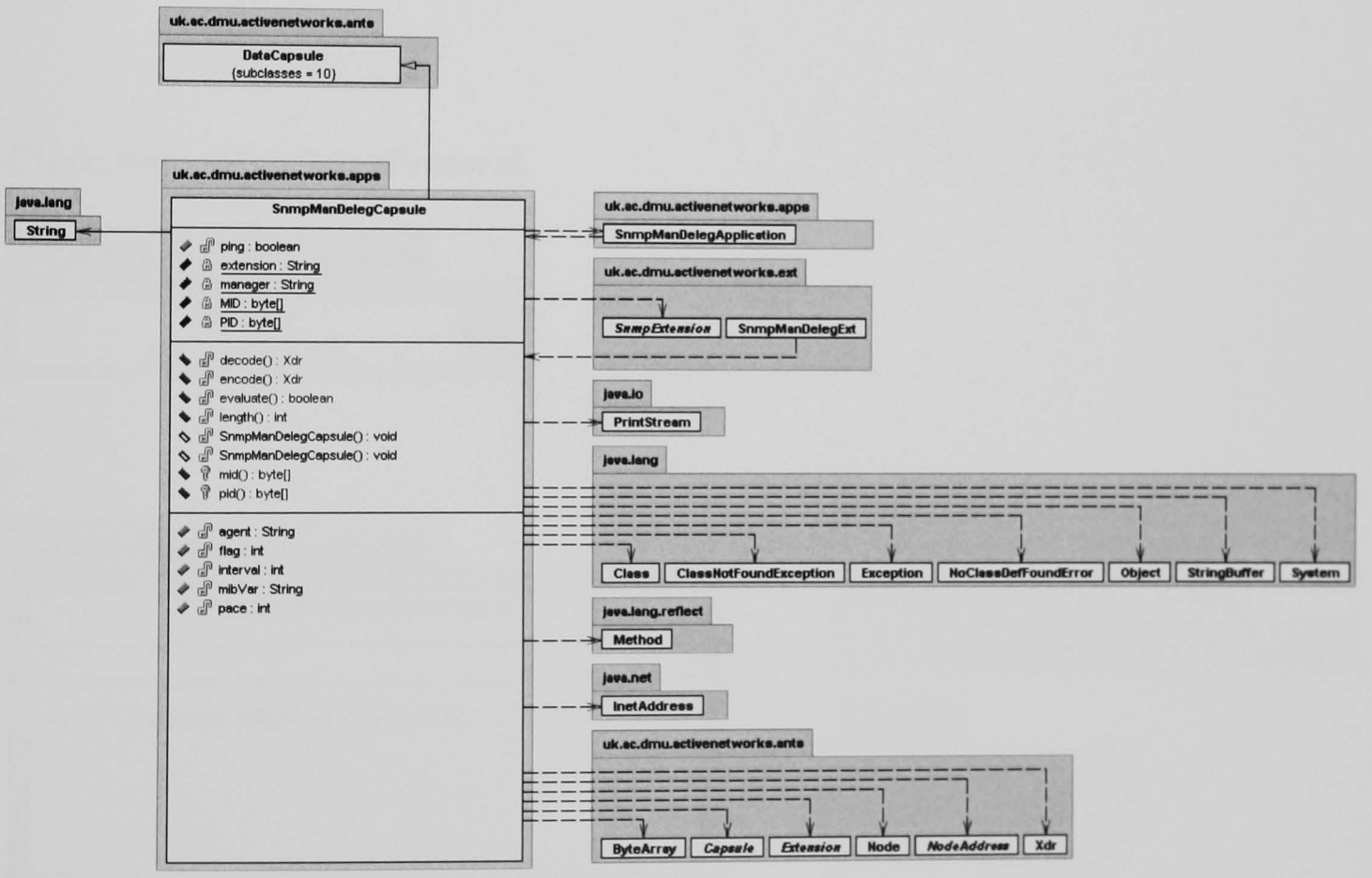
Class: SnmpManDelegApplication



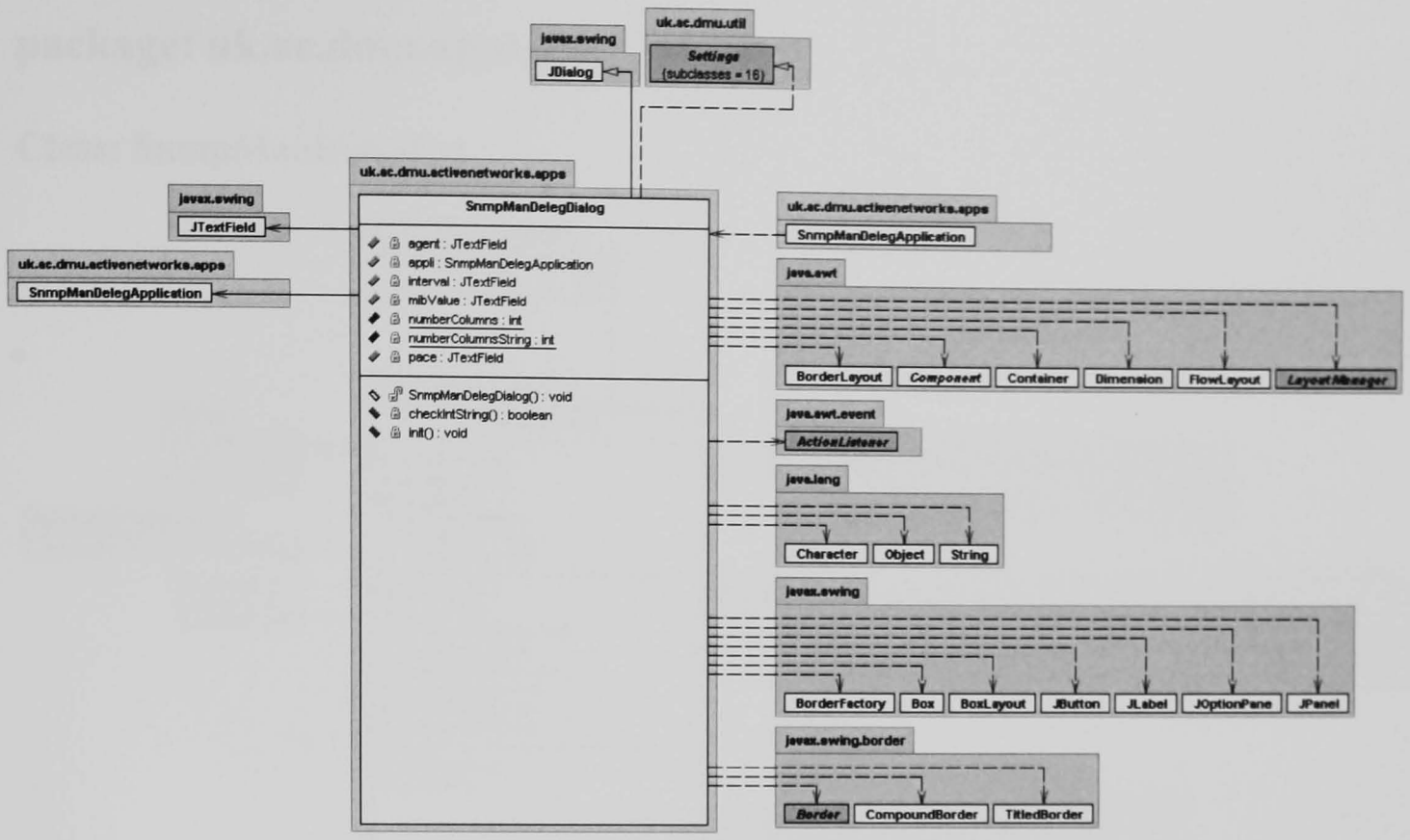
Class SnmpManDelegCapsule



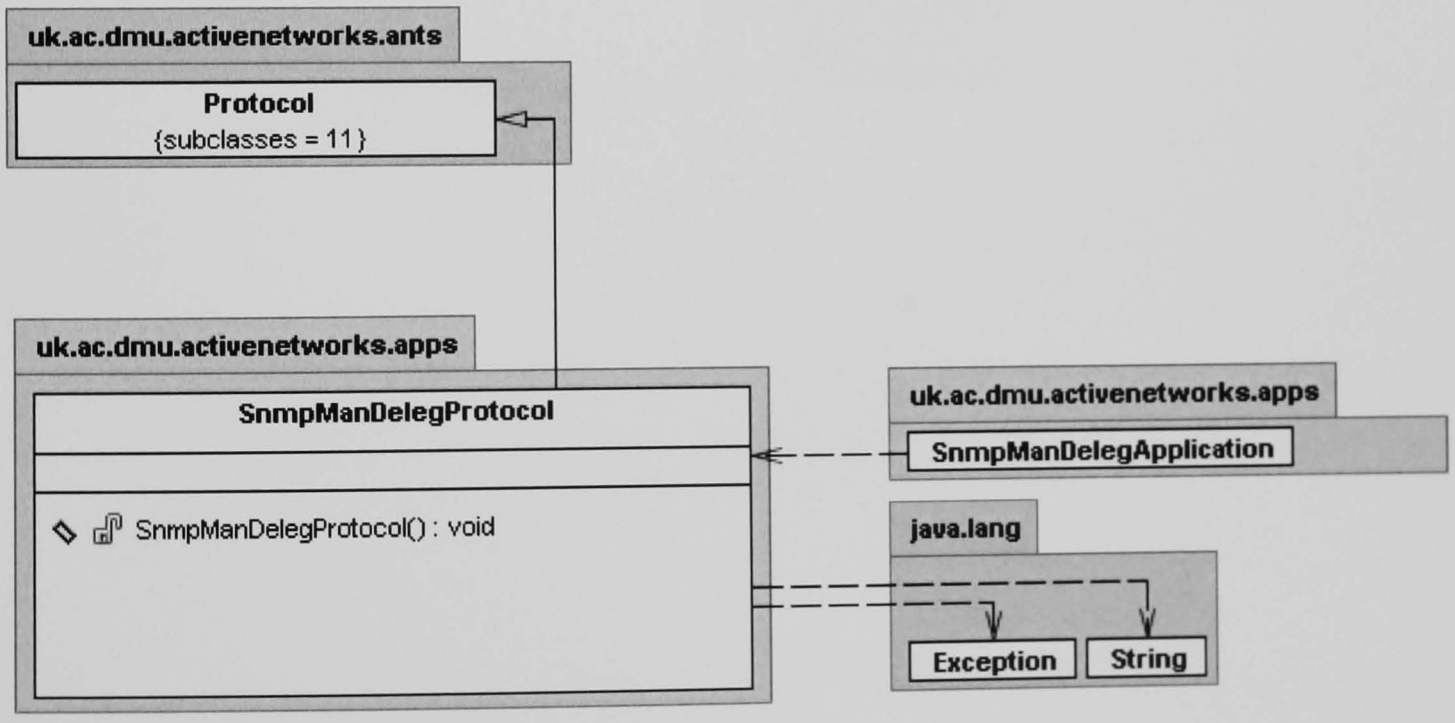
Class: SnmpManDelegCapsule



Class:SnmpManDelegDialog



Class: SnmpManDelegProtocol



package: uk.ac.dmu.apps.ext

Class: SnmpManDelegExt